Automatische a priori Fehlerabschätzungen zur Entwicklung optimaler Algorithmen und Intervallfunktionen in C-XSC

Frithjof Blomquist
Bergische Universität Wuppertal
November, 2005

Automatische a priori Fehlerabschätzungen zur Entwicklung optimaler Algorithmen und Intervallfunktionen in C-XSC

mit vielen Beispielen, Ergebnistabellen und Programm–Quelltexten.

Frithjof Blomquist
Bergische Universität Wuppertal
November, 2005

©**2005** Dr. Frithjof Blomquist, Adlerweg 6
66346 Püttlingen 3
E-Mail Adresse: Blomquist@math.uni-wuppertal.de

Text, Abbildungen, Tabellen und Programme wurden mit größter Sorgfalt erarbeitet. Der Autor kann jedoch für eventuelle fehlerhafte Angaben und deren Folgen weder eine juristische Verantwortung noch irgendeine Haftung übernehmen.
Das vorliegende Buch ist urheberrechtlich geschützt. Für nichtkommerzielle Zwecke darf es jedoch beliebig vervielfältigt und weitergegeben werden. Die Weitergabe dieses Buches in elektronischer oder gedruckter Form darf also in keinem Fall mit der Erhebung irgendwelcher Gebühren verbunden werden.

Der Postscriptfile dieses Buches und die Quellen der Hilfs- und Beispielprogramme stehen im Netz zur Verfügung unter:

`http://www.math.uni-wuppertal.de/~xsc/literatur/cxsc_docu.html`

Meiner Frau

Vorwort

Zweifellos hat sich die Datenverarbeitung vor allem in den letzten drei Jahrzehnten explosionsartig entwickelt und damit auch die Gesellschaft nachhaltig beeinflusst. In den Industrieländern kann man heute mit einem durchschnittlichen Monatsgehalt eine komplette Rechenanlage erwerben, und der Anwender kann damit eine Vielzahl von Problemfeldern erfolgreich bearbeiten. Vor ca. 30 Jahren war diese Entwicklung nicht abzusehen, und nach dieser Erfahrung ist es nahezu unmöglich, entsprechende Vorhersagen für die nächsten Jahrzehnte zu treffen.

Betrachtet man jedoch die bisherigen Möglichkeiten der Hardware aus Sicht des Mathematikers, so ergeben sich allerdings noch gewaltige Defizite. Um dies einzusehen, müssen wir die auf den Rechnern vorgegebenen Zahlensysteme genauer betrachten. Da sich die vier Grundoperationen in einem binären Format am schnellsten realisieren lassen, benutzt man für numerische Rechnungen meist ein Gleitkommasystem, das durch den weit verbreiteten IEEE–Standard definiert ist. Ein solches Gleitkommasystem besitzt jedoch noch entscheidende Nachteile:

Wegen des endlichen Exponentenbereichs gibt es z.B. für positive Zahlen einen kleinsten und einen größten Wert. Die größte Einschränkung ist jedoch bedingt durch die notwendigerweise endliche Mantissenbreite der Gleitkommazahlen. Daher stehen für numerische Rechnungen zwischen der kleinsten und größten positiven Zahl nur *endlich* viele Zahlen zur Verfügung, und zwischen zwei benachbarten Rasterzahlen gibt es ein ganzes Intervall mit positivem Durchmesser, dessen Zahlenkontinuum vom Rechner nicht erfasst wird. Der IEEE–Standard schreibt daher vor, dass das exakte Ergebnis einer der vier Grundoperationen zur jeweils nächsten Rasterzahl zu runden ist. Diese zunächst als gering einzuschätzenden Fehler sind aber der Grund für all die bekannten Probleme, die durch den Einsatz eines Computers bei numerischen Anwendungen auftreten können.

Im folgenden Beispiel betrachten wir der Einfachheit halber ein dezimales Datenformat mit nur fünf Mantissenstellen. In diesem Format liefert schon die Ausführung nur zweier Grundoperationen mit $a = 5.5555,\ b = 30.864$ an Stelle des exakten Wertes $a \cdot a - b = -4.1975 \cdot 10^{-4}$ den völlig falschen Maschinenwert $a \odot a \ominus b = 0$, da das exakte Produkt $a \cdot a = 30.86358025$ im angenommenen 5-stelligen Format zur

Rasterzahl $a \odot a = 30.864$ aufgerundet werden muss, und genau diese unvermeidbare Rundung macht die Berechnung des exakten Ergebnisses unmöglich! Jetzt kann man zwar einwenden, dass der Term $a \cdot a - b = -4.1975 \cdot 10^{-4}$ mit einer geeigneten Langzahl-Gleitkomma-Arithmetik sogar exakt ausgewertet werden kann, aber das geschilderte Problem ist grundsätzlicher Art, d.h. zu jeder Langzahlarithmetik kann man immer wieder Beispiele angeben, für die der Rechner völlig falsche Ergebnisse liefert!

Da in umfangreichen Algorithmen mit Millionen von Rechenoperationen das Auftreten obiger Beispiele nicht immer ausgeschlossen werden kann, war der Rechnereinsatz auf Basis einer Gleitkomma–Arithmetik für eine *zuverlässige* Numerik völlig unzureichend.

Zur Lösung der geschilderten Probleme wurde dann in den siebziger Jahren am Institut für Angewandte Mathematik der Universität Karlsruhe unter Leitung von Prof. Dr. U. Kulisch ein Festkommaspeicher in Software realisiert, in dem alle IEEE–Zahlen und alle Produkte aus solchen IEEE–Zahlen in beliebiger Anzahl *rundungsfehlerfrei* addiert werden können. Erst das Auslesen des Endergebnisses aus diesem Akkumulator in das benutzte IEEE–Format liefert dann einen maximalen Fehler von einem ulp (one **u**nit on **l**ast **p**osition). Nach der Implementierung dieses Software–Akkumulators in verschiedenen Hochsprachen, wie PASCAL-XSC, C-XSC oder FORTRAN-XSC war damit die Grundlage für eine zuverlässige Numerik mit dem Computer gegeben. Diese durch den Akkumulator erweiterte Gleitkomma–Arithmetik ist jedoch in Softwareausführung recht zeitaufwendig, und so wurde in den neunziger Jahren eine von der CPU getrennte Hardware–Lösung realisiert, die das folgende überraschende Ergebnis lieferte:

> In den Hardware–Akkumulator werden Produkte aus IEEE–Zahlen nicht nur rundungsfehlerfrei sondern im Vergleich zu den üblichen IEEE-Multiplikationen sogar deutlich schneller addiert!

Aus Sicht des Mathematikers ist es daher eine Katastrophe, dass bis zum heutigen Tage noch auf keiner CPU neben den Gleitkommaoperationen wenigstens ein Festpunkt–Akkumulator realisiert wurde. Mit einem solchen Prozessor könnten viele Algorithmen einfacher programmiert werden, und dem Anwender stünden bei deutlich kürzerer Laufzeit wirklich zuverlässige numerische Ergebnisse zur Verfügung. Neben dem geschilderten Hardware–Defizit gibt es aber noch einen weiteren, entscheidenden Schwachpunkt in der computergestützten Numerik:

Bei der Rechnerauswertung eines Algorithmus erhält man i.a. einen fehlerbehafteten Maschinenwert $\widetilde{y} \approx y$, wobei der exakte Wert y in vielen Fällen nicht bekannt ist, so dass eine nachträgliche Abschätzung des Fehlers praktisch unmöglich ist. Um jedoch mathematische Aussagen bezüglich des exakten (unbekannten) Wertes y zu erhalten, ist die Kenntnis einer Oberschranke des absoluten oder relativen Fehlers

unbedingt notwendig, denn nur bei Kenntnis etwa des absoluten Höchstfehlers Δ kann mit Hilfe des Maschinenwertes $\widetilde{y} = y + \delta,\ |\delta| \leq \Delta$ eine garantierte Unter- und Oberschranke des exakten Wertes y angegeben werden:

$$\alpha := \widetilde{y} - \Delta \leq y \leq \widetilde{y} + \Delta =: \beta$$

Mit der abgerundeten Unterschranke α hat man dann die mathematisch gesicherte Aussage $\alpha \leq y$ gewonnen nur mit den fehlerbehafteten Maschinenwerten $\widetilde{y}$ und der absoluten Fehlerschranke Δ[1]. Leider werden bis heute zu den meisten Algorithmen keine garantierten Fehlerschranken geliefert, so dass aus den fehlerbehafteten Maschinenwerten keine mathematischen Aussagen bez. der exakten Werte y möglich sind. So werden z.B. Fehlerschranken für die auf Coprozessoren realisierten Standardfunktionen nicht angegeben. Auf dieser Basis bleibt der Rechner für die Numeriker zwar ein sehr schnelles aber doch unzuverlässiges Hilfsmittel.

Erstaunlicherweise haben sich die meisten Mathematiker mit dieser völlig unbefriedigenden Situation abgefunden, denn in den meisten Fällen benutzt man den fehlerbehafteten Maschinenwert $\widetilde{y}$ in der Hoffnung, dass die erzielte Genauigkeit ausreichend sein wird. Mit der exakten Arbeitsweise eines Mathematikers ist diese Vorgehensweise natürlich nicht vereinbar! Die Entwicklung einer mathematisch zuverlässigen Numerik–Software muss nach folgenden Schema erfolgen:

1. Formulierung des Problems und Entwurf eines ersten Algorithmus

2. Berechnung einer garantierten Oberschranke des absoluten oder relativen Höchstfehlers

3. Ist die berechnete Oberschranke noch zu groß, so ist der Algorithmus weiter zu optimieren, bis eine vorgegebene Oberschranke unterschritten wird.

Erst zusammen mit einer solchen garantierten Oberschranke kann man dann mit Hilfe der i.a. fehlerbehafteten Computerergebnisse mathematisch gesicherte Aussagen über das meist unbekannte, exakte Ergebnis formulieren.

Wenn man in den kommenden Jahren die Hardware durch Realisierung eines oder mehrerer Festkommaspeicher zusammen mit einer Intervall–Arithmetik entscheidend verbessert haben wird und wenn zu den Algorithmen garantierte Fehlerschranken mitgeliefert werden, so kann man vermuten, dass die Mathematik als Basis wissenschaftlicher Anwendungen mit Hilfe des Computers ganz entscheidende Impulse erhalten wird. An dieser Stelle soll noch einmal ausdrücklich betont werden, dass die dazu notwendigen Ideen vom Karlsruher Institut für Angewandte Mathematik ausgegangen sind und mittlerweile weltweite Anerkennung gefunden haben.

[1] Da bei selbstverifizierenden Algorithmen die Schranken α, β mit $\alpha \leq y \leq \beta$ automatisch geliefert werden, erübrigt sich hier die Berechnung der Fehlerschranke Δ.

Es bleibt allerdings zu befürchten, dass die Hardware–Realisierung einer intelligenten Arithmetik zusammen mit mehreren Festkommaspeichern und die Entwicklung der dazu passenden Numeriksoftware mangels der dazu notwendigen finanziellen Mittel und wegen der oft fehlenden Fachkompetenzen in den entscheidenden Gremien wieder einmal im Ausland stattfinden wird, verbunden mit dem Verlust weiterer hochqualifizierter Arbeitsplätze. Welchen Segen hätte man der Wissenschaft in Deutschland erbringen können, wenn z.B. nur die Milliardenbeträge, die sinnlos für die nachträgliche Verengung bereits existierender breiter Straßen verschleudert wurden, gezielt in leistungsfähige Schulen und Universitäten investiert hätte!

Im Hinblick auf die geschilderten Anforderungen an eine mathematisch zuverlässige Software besteht nun die Aufgabe dieses Buches darin, dem Anwender alle diejenigen Werkzeuge zu vermitteln, die notwendig sind, um garantierte Fehlerschranken selbstentwickelter Algorithmen berechnen zu können; dabei habe ich mehrere Ziele verfolgt:

1. Der Leser soll in den ersten Kapiteln eine Übersicht erhalten über das verwendete Fehlerkalkül, dessen Grundideen bereits in den Dissertationen von Walter Krämer und Klaus Dieter Braune entwickelt wurden.

2. Zu einem gegebenen arithmetischen Ausdruck, der aus den vier Grundoperationen und den Standardfunktionen aufgebaut ist, kann der absolute oder relative Fehler mit Hilfe einfacher Programme abgeschätzt werden. Die dabei verwendeten **C-XSC** Funktionen stehen dem Anwender in einer eigenen Bibliothek zur Verfügung und werden in zahlreichen Beispielen erklärt.

3. Für spezielle Terme, wie z.B. $T(x) = 1 + x$, wurden Funktionen entwickelt, mit denen die bei der Auswertung von $T(x)$ unvermeidbaren Rundungsfehler $(1 \oplus x) - (1 + x)$ optimal abgeschätzt werden können. Da die Berechnung der Fehlerschranken stets mit Hilfe der in **C-XSC** implementierten Intervallarithmetik erfolgt, muss ein vorgegebenes Argumentintervall in hinreichend viele Teilintervalle zerlegt werden, um die bekannten Intervallüberschätzungen möglichst klein zu halten. In vielen Anwendungsbeispielen lernt der Anwender, dass die vorhandenen Werkzeuge zur Fehlerabschätzung mit großer Sorgfalt zu benutzen sind, um möglichst kleine Fehlerschranken zu erhalten.

4. Der Leser des Buches soll lernen, dass die Entwicklung eines optimalen Algorithmus zur Auswertung eines gegebenen Terms $T(x)$ nach folgendem Schema erfolgen muss:

   - Formuliere einen ersten Algorithmus und berechne zu einem gegebenen Argumentintervall $[a, b]$ eine möglichst kleine Fehlerschranke.

- Ist diese Fehlerschranke zu groß, so muss in verschiedene Teilintervallen von $[a, b]$ der jeweilige Algorithmus so abgeändert werden, dass die Fehlerabschätzungen in den neuen Teilintervallen hinreichend kleine Oberschranken liefern. Dabei ist darauf zu achten, dass die Laufzeiten der verwendeten Algorithmen nicht zu groß werden.

5. Mit den vielen Beispielen soll der Leser die notwendige Sicherheit erhalten, den Stoff richtig verstanden zu haben und die Programme korrekt anwenden zu können. Nach dem Studium dieses Buches sollte der Leser außerdem in der Lage sein, zu eigenen Aufgabenstellungen einen möglichst schnellen Algorithmus mit einer möglichst kleinen relativen Fehlerschranke selbstständig entwickeln zu können.

Als weitere Schwerpunkte sind zu nennen der Abschnitt 7.3, in dem gezeigt wird, wie man bei rationaler Approximation garantierte Fehlerschranken berechnen kann. Das Kapitel 6 beschäftigt sich mit Fehlern, die entstehen, wenn man Standardfunktionen mit Argumenten aufruft, die selbst wieder fehlerbehaftet sind.

Eine detaillierte Übersicht des behandelten Stoffes findet man im nachfolgenden Inhaltsverzeichnis und im Stichwortverzeichnis am Ende dieses Buches. Die vielen Programmbeispiele stehen im Netz als Quelltexte zur Verfügung und können in einer installierten **C-XSC** Umgebung problemlos übersetzt werden.

Zum Verständnis dieses Buches sollte der Leser die folgenden Voraussetzungen mitbringen:

1. Grundkenntnisse in **C++**zum Verständnis der Beispiele
2. Eine installierte **C-XSC** Version zur Auswertung der Routinen und Grundkenntnisse in **C-XSC** [7],[8],[14],[16];
3. Grundkenntnisse der Intervall–Arithmetik, [2]
4. Mathematik–Kenntnisse der Grundvorlesungen in Analysis

An dieser Stelle dürfen die vielfältigen Hilfen, die mir die Mitarbeiter des Instituts für Angewandte Mathematik der Universität Karlsruhe und der Bergischen Universität Wuppertal über die Jahre hinweg gewährt haben, nicht unerwähnt bleiben. Meinen herzlichen Dank richte ich dabei insbesondere an die Herren Ulrich Kulisch, Walter Krämer und Werner Hofschuster, ohne deren freundliche Unterstützung dieses Buch nicht zustandegekommen wäre. In zahllosen persönlichen Gesprächen fand ich immer wieder wertvolle Anregungen und damit auch die Kraft, dieses recht umfangreich gewordene Buch jetzt zu einem vorläufigen Abschluss zu bringen.

Frithjof Blomquist im November 2005.

Inhaltsverzeichnis

1 Begriffe und Bezeichnungen **1**
1.1 Gleitkommasysteme 1
1.2 IEEE-double Format 4
1.2.1 Eigenschaften 4
1.2.2 Vorgänger und Nachfolger 7
1.2.2.1 Einschließung von $\ln(1+x)$ 8
1.3 Speicherung von **IEEE**-Konstanten 11
1.4 Absolute und relative Fehler 13
1.5 Maximal genaue und hochgenaue Arithmetik 14

2 Fehlerschranken für $\{\oplus, \ominus, \odot, \oslash\}$ **17**
2.1 Ein Beispiel 17
2.2 Intervalloperanden 22
2.3 Absolute Fehlerschranken 24
2.3.1 Abschätzung des fortgepflanzten Datenfehlers 24
2.3.2 Abschätzung des Rundungsfehlers 25
2.3.3 Abschätzung des Gesamtfehlers 26
2.4 Relative Fehlerschranken 26
2.4.1 Abschätzung des fortgepflanzten Datenfehlers 27
2.4.2 Abschätzung des Rundungsfehlers 28
2.4.3 Abschätzung des Gesamtfehlers 29
2.5 Berechnung der Fehlerschranken 30
2.5.1 Maximal genaue Arithmetik 30
2.5.2 Hochgenaue Arithmetik 36
2.5.3 Zusammenstellung der Funktionen 39
2.5.4 Beispiel 42

3 Verbesserte Fehlerschranken **49**
3.1 Rundungsfehlerfreie Operationen 50
3.1.1 Beispiele 52

3.2 Spezielle Operanden 57
3.2.1 $T(x) = 1 + x$ 57
3.2.2 $T(x) = 1 - x$ 70
3.2.3 $T(x) = x - 1$ 71
3.2.4 Implementierung einer Intervallfunktion 72
3.3 Anwendungen 84
3.3.1 $T(x) = 1 + x^2$ 84
3.3.2 $T(x) = 1 + \frac{1}{4} \cdot x^2$ 87
3.3.3 $T(x) = 1 + \frac{2}{3} \cdot x^4$ 88

4 Auswertefehler **91**
4.1 Problemstellung 91
4.2 Polynome mit rationalen Koeffizienten 95
4.2.1 Hochgenaue Arithmetik 95
4.2.2 Beispiele 100
4.3 Polynome mit binären Koeffizienten 105
4.3.1 Beispiele 107
4.4 Rationale Funktionen 112
4.4.1 Beispiele 115
4.5 Spezielle Terme 124
4.5.1 $x + \frac{1}{2} x^2$ 124
4.5.1.1 $T_1(x) = x \cdot (1 + \frac{1}{2} x)$ 126
4.5.1.2 $T_2(x) = x + \frac{1}{2} x^2$ 127
4.5.1.3 $T_3(x) = x + \frac{1}{2} x^2,\ x = u + v$ 128
4.5.1.4 $T_4(x) = x + \frac{1}{2} x^2,\ x = u + v$ 131
4.5.2 $x + \frac{1}{2} x^2 + \frac{1}{6} x^3$ 135
4.5.2.1 $T_1(x) = x + (y + (q + z))$ 136
4.5.2.2 $T_2(x) = (u + y) + (q + (v + z))$ 138

5 Ein einfaches, nicht optimiertes Fehlerkalkül **141**
5.1 Problemstellung 141
5.2 Funktionen aus `abs_relh` zur Abschätzung rel. Fehler 142
5.3 Anwendungsbeispiele 143

6 Fehlerbehaftete Funktionsargumente **147**
6.1 Allgemeine Fehlerabschätzung 148
6.2 Fehlerschranken für spezielle Standardfunktionen 151
6.2.1 Die Funktion `sqrt`$(x) \approx \sqrt{x}$ 151
6.2.1.1 Ein Beispiel 154
6.2.2 Die Funktion `exp`$(x) \approx e^x$ 155
6.2.2.1 Beispiele 158

6.2.3 Die Funktion `ln`$(x) \approx \ln(x)$ . . . 161
6.2.4 Beispiele . . . 164
6.2.5 Die Funktion `lnp1`$(x) \approx \ln(1+x)$ . . . 167

7 Approximationsfehler **185**
7.1 Abschätzung des Gesamtfehlers . . . 185
7.1.1 $T(x) := h(x) \cdot f(x),\ f(x) \approx g(x)$ . . . 186
7.2 Approximation mit Taylorpolynomen . . . 188
7.2.1 Abschätzung mit der geometrischen Reihe . . . 189
7.2.2 Abschätzung mit dem Leibniz-Kriterium . . . 189
7.3 Rationale Approximation . . . 190
7.3.1 Numerische Auswertung der Schranken . . . 192
7.3.2 Das Programm `Approx-Error` . . . 196
7.3.3 Beispiele . . . 199
7.3.3.1 $f(x) = e^x$ . . . 199
7.3.3.2 $f(x) = \mathrm{erf}(x)$ . . . 201

8 Spezielle Standardfunktionen **211**
8.1 $\sqrt{x+1}-1$ . . . 211
8.1.1 Punktargumente . . . 211
8.1.2 Intervallargumente . . . 222
8.2 $\ln(\sqrt{x^2+y^2})$ . . . 227
8.2.1 Punktargumente . . . 227
8.2.2 Intervallargumente . . . 285
8.2.3 Numerische Ergebnisse . . . 286
8.2.4 Quelltext für Punkt- und Intervallargumente . . . 288
8.3 $\sqrt{x^2-1}$ . . . 294
8.3.1 Punktargumente . . . 294
8.3.2 Intervallargumente . . . 310
8.3.3 Numerische Ergebnisse . . . 311
8.3.4 Quelltext für Punkt- und Intervallargumente . . . 312
8.4 $\sqrt{1-x^2}$ . . . 315
8.4.1 Punktargumente . . . 315
8.4.2 Intervallargumente . . . 323
8.4.3 Numerische Ergebnisse . . . 323
8.4.4 Quelltext für Punkt- und Intervallargumente . . . 325
8.5 e^{-x^2} . . . 327
8.5.1 Punktargumente . . . 328
8.5.2 Intervallargumente . . . 336
8.5.3 Numerische Ergebnisse . . . 337

8.5.4 Quelltext für Punkt- und Intervallargumente 339
8.6 arcosh$(1+x)$ 342
8.6.1 Algorithmen für arcosh$(1+x)$ 342
8.6.2 Abschätzung der Approximationsfehler 344
8.6.3 Punktargumente 346
8.6.3.1 Fehlerabschätzung in A_1 346
8.6.3.2 Fehlerabschätzung in A_2 346
8.6.3.3 Fehlerabschätzung in A_3 349
8.6.3.4 Fehlerabschätzung in A_4 350
8.6.3.5 Fehlerabschätzung in A_5 352
8.6.3.6 Fehlerabschätzung in A_6 352
8.6.4 Intervallargumente 354
8.6.5 Numerische Ergebnisse 355
8.6.6 Quelltext für Punkt- und Intervallargumente 356

9 Staggered Arithmetik **359**
9.1 Implementierung in **C++** 359
9.1.1 Variablen vom Typ *l_real* 359
9.1.2 Variablen vom Typ *l_interval* 363
9.2 Anmerkungen zur Fehlerabschätzung 365
9.3 coth(x) 369
9.3.1 Der Algorithmus 370
9.3.2 Fehlerabschätzung in A_1 371
9.3.3 Fehlerabschätzung in A_4 373
9.3.4 Numerische Ergebnisse 374
9.3.5 Quelltext 377
9.4 $\sqrt{1-x^2}$ 380
9.4.1 Der Algorithmus 380
9.4.2 Numerische Ergebnisse 381
9.4.3 Quelltext 382

10 Minimumeinschließung konvexer Funktionen **385**
10.1 Problemstellung 385
10.2 Einschließung der Minimumstelle x_0 386
10.3 Einschließung des Minimums $f(x_0)$ 387
10.4 Anwendungen 388
10.4.1 $f(x) = x^2 \cdot \ln(x),\ x > 0$ 389
10.4.2 $f(x) = e^{-x} \cdot \ln(1+x),\ x > -1$ 391

A Das Programm `Approx-Error` **395**

B Das Programm `Min-Incl` 407

C Fehlerschranken der Standardfunktionen 415

Kapitel 1

Begriffe und Bezeichnungen

1.1 Gleitkommasysteme

Da jeder Computer nur mit einem endlichen Arbeitsspeicher ausgestattet ist, stehen für numerische Rechnungen nur endlich viele Rasterzahlen mit endlicher Mantissenlänge zur Verfügung. Dies hat zur Folge, dass schon so einfache Zahlen wie 1/3 oder 1/7 i.a. nicht exakt gespeichert werden können. Aus dem gleichen Grund kann das Produkt zweier Maschinenzahlen im Rechner i.a. nicht exakt dargestellt werden, und das bestmögliche Ergebnis erhält man nur, wenn garantiert wird, dass das exakte Produkt zur nächstgelegenen Rasterzahl gerundet wird. Bedingt durch diese unvermeidbaren Rundungen kann das Ergebnis einer Rechnerauswertung schon nach wenigen Operationen völlig unbrauchbar werden. Als Beispiel betrachten wir den sehr einfachen Term

$$\mathrm{T}(x, y) := 9x^4 - y^4 + 2y^2$$

Die Auswertung für $x = 10864$ und $y = 18817$ mit dem HP 15C-Taschenrechner liefert an Stelle des exakten Ergebnisses $\mathrm{T}(10864, 18817) = 1$ den völlig falschen Wert $8.15\ldots\cdot 10^6$, und ein Taschenrechner mit einem anderen Zahlenformat wird i.a. ein anderes aber ebenfalls völlig unbrauchbares Ergebnis liefern. Das obige Beispiel zeigt:

1. Rechnerergebnisse können auch bei nur wenigen Operationen nicht nur auf den letzten Stellen fehlerhaft sein!

2. Die Fehler sind wesentlich auch vom benutzten Zahlensystem abhängig.

In diesem Buch werden Möglichkeiten aufgezeigt, zu einem vorgegebenen Algorithmus eine a priori Fehlerabschätzung durchzuführen. Daher müssen wir das benutzte Zahlensystem genauer betrachten! Die speichereffizientesten Zahlenformate sind

Gleitkomma-Systeme (floating-point system), die wir genauer betrachten wollen. Zur Erleichterung beschränken wir uns in den folgenden Beispielen auf das für uns leicht lesbare Dezimalsystem. Für jede andere Basis $B >= 2$ ergeben sich ganz entsprechende Überlegungen.
In der Mathematik schreibt man gewöhnlich eine reelle Zahl $x \in I\!R$ im Dezimalsystem in der Form:

$$x = \pm d_n d_{n-1} \ldots d_1 d_0 \centerdot d_{-1} d_{-2} \ldots$$

dabei steht d_i mit $i = n, n-1, \ldots$ stellvertretend für je eine der Ziffern $0, 1, 2, \ldots, 9$, und die durch den Dezimalpunkt getrennte Ziffernfolge hat die Bedeutung:

$$x = d_n \cdot 10^n + \ldots + d_1 \cdot 10^1 + d_0 \cdot 10^0 + d_{-1} \cdot 10^{-1} + d_{-2} \cdot 10^{-2} + \ldots$$

Beim praktischen Rechnen können wegen des endlichen Arbeitsspeichers nur Zahlen mit endlich vielen Ziffern verwendet werden. Die Teilmenge dieser endlichen Dezimalzahlen bezeichnen wir mit $\mathbf{S}(B, k)$, wobei B die Basis (eine ganze Zahl ≥ 2, in unseren Beispielen 10) und k die Anzahl der dargestellten Ziffern (eine ganze Zahl ≥ 1) zu dieser Basis festlegt. Bei konventionellen Gleitkommasystemen sind nur die Ziffern 0 bis $B - 1$ zugelassen. Ganz analog zu unserem dezimalen Beispiel hat die Zahl

$$x = \pm d_n d_{n-1} \ldots d_1 d_0 \centerdot d_{-1} d_{-2} \ldots d_{n-k+1} \qquad \text{den Wert}$$

$$x = \sum_{i=n-k+1}^{n} d_i \cdot b^i$$

Betrachten wir nun eine beliebige Dezimalzahl, z.B. 3.14159, so gibt es neben der angegebenen Darstellung noch beliebig viele andere Möglichkeiten, die sich durch die Position des Dezimalpunkts unterscheiden:

$$3.14159 = 0.314159 \cdot 10^1 = 0.0314159 \cdot 10^2 = 0.00314159 \cdot 10^3 = ...$$

Um die Darstellung eindeutig festzulegen, führt man eine Normierung ein. Wir bezeichnen die Darstellung einer Zahl $x \neq 0$ in der Form

$$x = \pm m \cdot B^e \in \mathbf{S}(B, k) \quad \text{mit} \quad 1 \leq m < B \quad \text{und einer ganzen Zahl } e$$

als **normalisierte** Gleitkommadarstellung. m und e heißen Mantisse und Exponent von x. Das Vorzeichen haben wir durch $\pm$ gekennzeichnet. Bei dieser Darstellung wird der Gleitpunkt stets hinter die erste signifikante Ziffer gesetzt und der Faktor B^e entsprechend angepasst. Das obige Beispiel besitzt damit die normalisierte Form:

$$3.14159 \cdot 10^0$$

Bei der Darstellung einer normalisierten Gleitkommazahl auf einem Computer steht nicht nur für die Mantisse sondern auch für den Exponenten jeweils eine feste Anzahl von Ziffern zur Verfügung. Diese Zahlen heißen Mantissen- bzw. Exponentenlänge und sind implementierungsabhängig. Wir bezeichnen diese endliche Teilmenge darstellbarer normalisierter Gleitkommazahlen mit $\mathbf{S}(B, k, e_1, e_2)$. Die ganzen Zahlen e_1, e_2 legen den Bereich für den Exponenten fest. Die kleinste bzw. größte positive Zahl in $\mathbf{S}(10, 5, -10, 10)$ lautet:

$$\texttt{MinReal} = 1.0000 \cdot 10^{-10} \quad \text{bzw.} \quad \texttt{MaxReal} = 9.9999 \cdot 10^{+10}$$

Der gesamte Wertebereich liegt also im Intervall $[-\texttt{MaxReal}, +\texttt{MaxReal}]$.

In unmittelbarer Nähe der Null ergibt sich zunächst eine Lücke, die in einem Teilbereich bei vielen Implementierungen mit denormalisierten Zahlen äquidistant aufgefüllt wird. Diese denormalisierten Zahlen sind gekennzeichnet durch den kleinsten Exponenten e_1 sowie durch mindestens eine führende Null in ihrer Mantisse. Sie besitzen daher eine geringere relative Genauigkeit, was bei Fehlerabschätzungen zu beachten ist.

Zusammenfassung:
Seien B, k, e, e_1, e_2 ganze Zahlen, mit:

$$B > 1 \text{ (Basis)}, \quad k > 0 \text{ (Mantissenlänge)} \quad \text{und} \quad e_1 \le e \text{ (=Exponent) } \le e_2.$$

Dann besteht das Gleitkommasystem

$$\mathbf{S}(B, k, e_1, e_2) \equiv \mathbf{S}(B, k) \tag{1.1}$$

aus der Zahl Null sowie aus allen rationalen Zahlen $x \neq 0$ der Form

$$x = \pm m \cdot B^e, \qquad e_1 \le e \le e_2, \qquad 1 \le m < B \tag{1.2}$$

$$\text{Mantisse:} \qquad m = \sum_{i=0}^{k-1} d_i \cdot B^{-i} = d_0 \,.\, d_1 d_2 \ldots d_{k-1}$$

$$d_0 \in \{1, 2, \ldots, B-1\}; \qquad d_i \in \{0, 1, \ldots, B-1\}, \;\; i = 1, 2, \ldots, k-1;$$

Eigenschaften:

- Anzahl Z der Elemente: $\qquad Z = 2 \cdot (B-1) \cdot B^{k-1}(e_2 - e_1 + 1) + 1$
- Durch die Forderung $d_0 > 0$ wird die Darstellung (1.2) eindeutig, und man spricht von **normalisierten** Zahlen;

- Die Zahl Null ist nicht in der Form (1.2) darstellbar; man verabredet daher:

$$0 = 0.\underbrace{000\ldots000}_{k-1\text{ Ziffern}} \cdot B^0$$

- x, y: aufeinanderfolgende Rasterzahlen mit gleichen Exponenten e;
  z.B.: $x = 1 \cdot B^0 \cdot B^e$; $y = (1 \cdot B^0 + 1 \cdot B^{1-k}) \cdot B^e \quad \rightarrow \quad y - x = B^{e+1-k}$,
  d.h. eine äquidistante Verteilung der Gleitpunktzahlen existiert nur bei Rasterzahlen mit gleichen Exponenten e.

- $x = 1 \cdot B^0 \cdot B^e$ ist **kleinste**, positive Rasterzahl zum Exponenten $e \quad \rightarrow$

- $\dfrac{y-x}{x} \leq B^{1-k}$: Maximum der relativen Gitterabstände positiver Zahlen;

1.2 IEEE-double Format

Mit Hilfe der Bezeichnungen des letzten Abschnitts wird das IEEE-double Format für normalisierte Zahlen nach (1.1) beschrieben durch [25]:

$$\mathbf{S}(2, 53, -1022, +1023) \equiv \mathbf{S}(2, 53)$$

Für eine normalisierte double-Zahl $\widetilde{x} \neq 0$ gilt:

$$\widetilde{x} = \pm m \cdot 2^e, \quad e_1 = -1022 \leq e \leq e_2 = +1023, \quad 1 \leq m < 2 \tag{1.3}$$

$$\text{Mantisse:} \qquad m = \sum_{i=0}^{52} d_i \cdot 2^{-i} = d_0 \,\mathbf{.}\, d_1 d_2 \ldots d_{52}$$

$$d_0 = 1; \qquad d_i \in \{0, 1\}, \;\; i = 1, 2, \ldots, 52;$$

1.2.1 Eigenschaften

- Durch die Forderung $d_0 = 1$ wird die Darstellung (1.3) eindeutig, und man spricht von **normalisierten** Zahlen;

- Die Zahl Null ist nicht in der Form (1.3) darstellbar; man verabredet daher:

$$0 = 0.\underbrace{000\ldots000}_{52\text{ Ziffern}} \cdot 2^0$$

- Sind $\widetilde{x}, \widetilde{y}$ aufeinanderfolgende Rasterzahlen mit gleichen Exponenten e, so gilt für den Gitterabstand: $\widetilde{y} - \widetilde{x} = 2^{e-52}$, d.h. eine äquidistante Verteilung der Gleitpunktzahlen existiert nur bei Rasterzahlen mit gleichen Exponenten e.
- $\widetilde{x} = 1 \cdot 2^e$ ist **kleinste**, positive Rasterzahl zum Exponenten e $\rightarrow$
- $\dfrac{\widetilde{y} - \widetilde{x}}{\widetilde{x}} \leq 2^{-52}$: Maximum der relativen Gitterabstände normalisierter Zahlen;
- Kleinste positive normalisierte double-Zahl:
  `MinReal` $= 1 \cdot 2^{-1022} = 2.22507385850720138\ldots \cdot 10^{-308}$
- Größte positive normalisiert double-Zahl:
  `MaxReal` $= 1.111\ldots11 \cdot 2^{+1023} = 1.79769313486231570\ldots \cdot 10^{+308}$

Der Bereich $[-$`MinReal`$, +$`Minreal`$]$ wird äquidistant durch denormalisierte Zahlen aufgefüllt, die alle den minimalen Exponenten $e_2 = -1022$ besitzen und deren Mantissen, im Gegensatz zu den normalisierten Zahlen, als signifikante Ziffer die Zahl $d_0 = 0$ besitzen. Die größte positive denormalisierte Zahl hat also die Form: $0.111\ldots11 \cdot 2^{-1022}$, und die kleinste positive denormalisierte Zahl `minreal` ist gegeben durch:
$0.00\ldots001 \cdot 2^{-1022} = 2^{-52} \cdot 2^{-1022} = 2^{-1074} = 4.9406564584124\ldots \cdot 10^{-324}$, und `minreal` $= 2^{-1074}$ ist dabei gleichzeitig auch der Abstand benachbarter denormalisierter Rasterzahlen.

Zur Speicherung einer double Zahl benutzt man 8 Byte mit 64 Bit. Die Zahl $\widetilde{x} = 0$

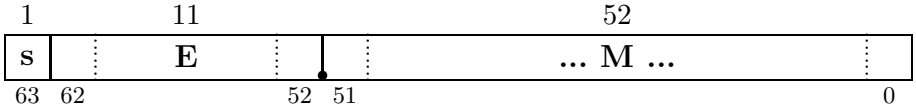


Abbildung 1.1: Speicherformat einer IEEE-double–Konstanten

wird gespeichert, indem alle 64 Bit auf Null gesetzt werden. Für $\widetilde{x} \neq 0$ bestimmt das 63.-te Bit das Vorzeichen von $\widetilde{x}$:

$$s = \begin{cases} 0 & \text{falls } \widetilde{x} > 0 \\ 1 & \text{falls } \widetilde{x} < 0 \end{cases}$$

Die vorzeichenlose integer–Zahl E wird definiert durch das 52. bis 62. Bit. Mit Hilfe des Exponenten e aus (1.3), mit $-1022 \leq e \leq +1023$, wird E für normalisierte Zahlen berechnet durch: $\mathrm{E} = e + 1023 \geq 1$.

Bei normalisierten Zahlen gilt für die führende Mantissenziffer d_0 stets $d_0 = 1$, so dass diese Ziffer nicht gespeichert werden muss (hidden bit). Die restlichen 52

Mantissenziffern legt man dann im 51. bis 0.-ten Bit ab. Die in (1.3) angegebene Mantisse m kann daher geschrieben werden als $m = 1.\mathrm{M}$, so dass eine normalisierte Zahl $\widetilde{x} \neq 0$ jetzt dargestellt werden kann als:

$$\widetilde{x} = \pm m \cdot 2^e = (-1)^s \cdot 1.\mathrm{M} \cdot 2^{\mathrm{E}-1023} \tag{1.4}$$

Für denormalisierte Zahlen gilt entsprechend

$$\widetilde{x} = (-1)^s \cdot 0.\mathrm{M} \cdot 2^{-1022}, \tag{1.5}$$

wobei jetzt E nur mit Nullen aufgefüllt wird, so dass nur $\widetilde{x} = 0$ und alle denormalisierten Zahlen durch $\mathrm{E} = 0$ gekennzeichnet sind.
Beispiel: Soll die Zahl -44 im double–Format gespeichert werden, so muss sie zunächst binär normalisiert werden zu $1.375_{dez} \cdot 2^5 = 1.011_{bin} \cdot 2^5$. Durch Addition von 1023 zum Exponenten $e = 5$ ergibt sich daraus:
$s = \mathbf{1}, \mathrm{E} = 1028_{dez} = 10000000100_{bin}$ und $\mathrm{M} = \mathbf{01100\ldots00}$. Daraus ergibt sich die gesuchte Bitfolge: $\mathbf{1}\,10000000100\,\mathbf{01100\ldots00}$ aus insgesamt 64 Bit.

In **C-XSC** existieren die beiden Funktionen

- `int expo(const real& x);`
- `real mant(const real& x);`

mit denen zu einer gegebenen Zahl `x` $\neq 0$ der Klasse `real` mit Hilfe der Anweisungen `int ex = expo(x); real mt = mant(x);` die Mantisse `mt` und der Exponent `ex` von `x` berechnet werden können. Es gilt:

$$\mathtt{x} = \mathtt{mt} \cdot 2^{\mathtt{ex}}, \qquad 0.5 \leq |\mathtt{mt}| < 1$$

Vergleicht man `mt` und `ex` mit den Größen m, e aus (1.3) von Seite 4, so bestehen die Beziehungen:

$$\mathtt{mt} = \frac{1}{2} \cdot m \qquad \text{und} \qquad \mathtt{ex} = e + 1$$

Für $x = 0$ erhält man: $\mathtt{mt} = 0$ und $\mathtt{ex} = -2147483647$.

Abschließend noch eine Bemerkung zu den **Konvertierungsfehlern**. Oft wird übersehen, dass die Werte eines dezimalen Gleitpunktsystems i.a. nicht exakt im double–System $\mathbf{S}(2, 53, -1022, +1023)$ darstellbar sind und umgekehrt. So hat z.B. die noch sehr einfache Dezimalzahl 0.1 eine nicht abbrechende Dualdarstellung der Form $0.0001100011\ldots$ und ist daher nicht in $\mathbf{S}(2, 53, -1022, +1023)$ enthalten. Wird dann der unendliche Dualbruch auf 53 Mantissenstellen gerundet, so ist der dabei auftretende Konvertierungsfehler bei späteren Fehlerabschätzungen natürlich mit zu berücksichtigen!

1.2.2 Vorgänger und Nachfolger

Wir betrachten double-Zahlen $\widetilde{x} \in S(2, 53)$ der Form $\widetilde{x} = m \cdot 2^{ex}$, mit $ex \in \mathbb{Z}$ und $-1022 \leq ex \leq +1023$. Die Mantisse m liegt für normalisierte und denormalisierte Zahlen in unterschiedlichen Bereichen:

Für normalisierte Zahlen gilt: $1 \leq |m| < 2$ und $-1022 \leq ex \leq +1023$.
Für denormalisierte Zahlen gilt: $0 < |m| < 1$ und $ex = -1022$.
Für die Zahl Null gilt: $m = 0$ und $ex = 0$.

Vorgänger und Nachfolger einer double-Zahl $\widetilde{x} \in S(2,53)$ sind die zu $\widetilde{x}$ nächstgelegenen Rasterzahlen, die in der **C-XSC** Umgebung mit Hilfe der Funktionen `pred()` und `succ()` berechnet werden. Es gilt: `pred`$(3) \equiv$ `pred`(3.0), `succ`$(2) \equiv$ `succ`(2.0).

Für $ex = 0$, d.h. $\widetilde{x} = 0$, oder $ex = -1022$ gelten die Beziehungen:

$$\texttt{pred}(\widetilde{x}) = \widetilde{x} - 2^{-1074}; \qquad \texttt{succ}(\widetilde{x}) = \widetilde{x} + 2^{-1074}; \tag{1.6}$$

Für $-1021 \leq ex \leq +1023$ und $ex \neq 0$ gilt mit $1 \leq |m| < 2$:

$$\texttt{pred}(\widetilde{x}) = \begin{cases} \widetilde{x} - 2^{ex-53} & \text{falls } m = 1 \\ \widetilde{x} - 2^{ex-52} & \text{sonst} \end{cases} \tag{1.7}$$

$$\texttt{succ}(\widetilde{x}) = \begin{cases} \widetilde{x} + 2^{ex-53} & \text{falls } m = -1 \\ \widetilde{x} + 2^{ex-52} & \text{sonst} \end{cases} \tag{1.8}$$

1.2.2.1 Einschließung von $\ln(1+x)$

Eine erste einfache Anwendung von (1.6), (1.7) ist die Berechnung einer garantierten Unterschranke von $\ln(1+x)$ in der Umgebung von $x = 0$. Wir werden danach zur Vervollständigung auch die Oberschranke für die gleich Funktion angeben. Hierzu sind die obigen Formeln für den Vorgänger und den Nachfolger einer double-Zahl jedoch nicht erforderlich.

Für $\widetilde{x} = m \cdot 2^{ex}$, mit $1 \leq m < 2$ gilt unter der Voraussetzung $-1021 \leq ex \leq -54$ mit (1.7) im ungünstigsten Fall $m = 1$ zur Berechnung einer Unterschranke die Abschätzung:

$$\texttt{pred}(\widetilde{x}) = \widetilde{x} - 2^{ex-53} \leq \ln(1+\widetilde{x}) \tag{1.9}$$

Zum Beweis der Ungleichung in (1.9) benutzen wir die für $x \geq 0$ gültige Beziehung: $x - \frac{x^2}{2} \leq \ln(x+1)$; damit ist dann für positive $\widetilde{x}$ zu zeigen:

$$\begin{aligned} & \widetilde{x} - 2^{ex-53} \leq \widetilde{x} - \frac{\widetilde{x}^2}{2}, \text{ falls } -1021 \leq ex \leq -54 \\ \iff \quad & 2^{ex-53} \geq \frac{\widetilde{x}^2}{2} = m^2 \cdot 2^{2\,ex-1} \\ \iff \quad & m^2 \cdot 2^{ex+52} \leq 1 \end{aligned}$$

Wegen $m^2 < 4$ bleibt damit zu zeigen $2^{ex+54} \leq 1$, und diese Ungleichung ist unter der obigen Voraussetzung $-1021 \leq ex \leq -54$ tatsächlich erfüllt.

Der Beweis von $\texttt{pred}(\widetilde{x}) \leq \ln(\widetilde{x}+1)$ muss jetzt für positive $\widetilde{x}$ noch geführt werden für den Exponenten $ex = -1022$. Nach (1.6) ist damit wieder zu zeigen:

$$\begin{aligned} & \widetilde{x} - 2^{-1074} \leq \widetilde{x} - \frac{\widetilde{x}^2}{2} \\ \iff \quad & -2^{-1074} \leq -\frac{\widetilde{x}^2}{2} \\ \iff \quad & \widetilde{x}^2 \leq 2 \cdot 2^{-1074} \\ \iff \quad & |\widetilde{x}| \leq \sqrt{2} \cdot 2^{-537}, \end{aligned}$$

und die letzte Ungleichung ist für $ex = -1022$ sicher erfüllt.

Für $ex = 0$, d.h. $\widetilde{x} = 0$ ist eine optimale Unterschranke von $\ln(\widetilde{x}+1)$ natürlich durch $\widetilde{x} = 0$ selbst gegeben, so dass jetzt eine Unterschranke für $\ln(\widetilde{x}+1)$ nur noch für negative $\widetilde{x} \in S(2,53)$, mit $-1 < \widetilde{x} < 0$ anzugeben ist.

Für $\widetilde{x} = m \cdot 2^{ex}$, mit $-2 < m \leq -1$ gilt unter der Voraussetzung $-1021 \leq ex \leq -55$ mit (1.7) die Abschätzung:

$$\texttt{pred}(\widetilde{x}) = \widetilde{x} - 2^{ex-52} \leq \ln(1+x) \tag{1.10}$$

Zum Beweis der Ungleichung in (1.10) benutzen wir die für $x > -1$ gültige Beziehung $\frac{x}{1+x} \leq \ln(x+1)$, damit ist dann für negative $\widetilde{x}$ zu zeigen:

$$\begin{aligned} & \widetilde{x} - 2^{ex-52} \leq \frac{\widetilde{x}}{\widetilde{x}+1}, \quad \text{falls } -1021 \leq ex \leq -55 \\ \iff \quad & 2^{ex-52} \geq \frac{-\widetilde{x}^2}{\widetilde{x}+1} \end{aligned}$$

Für[1] $-\frac{1}{2} < \widetilde{x} < 0$ gilt $\frac{1}{x+1} < 2$, so dass zu zeigen bleibt:

$$\begin{aligned} & 2^{ex-52} \geq 2 \cdot x^2 = 2 \cdot m^2 \cdot 2^{2ex} = m^2 \cdot 2^{2ex+1}, \text{ falls } -1021 \leq ex \leq -55 \\ \iff \quad & m^2 \cdot 2^{ex+53} \leq 1 \end{aligned}$$

Wegen $m^2 < 4$ bleibt damit zu zeigen: $2^{ex+55} \leq 1$, und diese Ungleichung ist für $-1021 \leq ex \leq -55$ tatsächlich erfüllt. Der Nachweis, dass $\texttt{pred}(\widetilde{x}) < \ln(1+\widetilde{x})$ für $ex = -1022$ auch für negative $\widetilde{x}$ gilt, kann wieder mit (1.6) geführt werden und bleibt dem Leser überlassen.

Damit sind wir jetzt in der Lage, für die Funktion $\ln(1+x)$ in der unmittelbaren Umgebung des Ursprungs garantierte Unterschranken anzugeben. Um diese Unterschranken für den ganzen Definitionsbereich formulieren zu können, greifen wir jetzt schon den Begriff der relativen Fehlerschranke auf, mit der auf Seite 78 Unterschranken und Oberschranken für monoton wachsende oder fallende Funktionen angegeben werden.

[1] $-\frac{1}{2} < \widetilde{x} < 0$ ist hier keine wirkliche Einschränkung, da die Berechnung einer Unterschranke mit $\texttt{pred}(\widetilde{x})$ nur in der unmittelbaren Umgebung des Ursprungs erfolgt.

Die von Null verschiedenen double-Zahlen $\widetilde{x}$ schreiben wir wie bisher in der Form $\widetilde{x} = m \cdot 2^{ex} \in S(2,53)$, dabei gilt:
$1 \leq |m| < 2$ und $-1022 \leq ex \leq +1023$ für den normalisierten Zahlenbereich und $0 < |m| < 1$ mit $ex = -1022$ für den denormalisierte Zahlenbereich.

Bedeutet $\varepsilon_0 = 2.5082 \cdot 10^{-16}$ die relative Fehlerschranke der in **C-XSC** implementierten Funktion $\texttt{lnp1}(\widetilde{x}) \approx \ln(1+\widetilde{x})$, so gilt nach (3.19) von Seite 78 für die Unterschranken der monoton wachsenden Funktion $\ln(1+x)$

$$\ln(1+\widetilde{x}) \geq 0, \text{ falls } \widetilde{x} = 0$$

Sei $\widetilde{x} > 0$, dann gilt:

$$\ln(1+\widetilde{x}) \geq \begin{cases} \texttt{pred}(\widetilde{x}), & \text{falls } ex \leq -54 \\ \texttt{lnp1}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon_0}{1+\varepsilon_0}\right) & \text{falls } ex > -54 \end{cases}$$

Sei $\widetilde{x} < 0$, dann gilt:

$$\ln(1+\widetilde{x}) \geq \begin{cases} \texttt{pred}(\widetilde{x}), & \text{falls } ex \leq -55 \\ \texttt{lnp1}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon_0}{1-\varepsilon_0}\right) & \text{falls } ex > -55 \end{cases}$$

Der Vollständigkeit halber sollen für die Funktion $\ln(1+x)$ jetzt noch geeignete Oberschranken angegeben werden, obwohl zu deren Berechnung die Funktionen `pred()` und `succ()` nicht benötigt werden.

Für $x > -1$ gilt zunächst stets $\ln(1+x) \leq x$, wobei jedoch die Oberschranke x nur in einer hinreichend kleinen Umgebung von Null günstig, d.h. hinreichend klein ist. Außerhalb dieser Umgebung kann nach (3.20) von Seite 78 eine weitere garantierte Oberschranke angegeben werden:

$$\ln(1+\widetilde{x}) \leq \begin{cases} \texttt{lnp1}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon_0}{1-\varepsilon_0}\right) < \texttt{lnp1}(\widetilde{x}) \odot (Z1p \oslash N1p) & \text{falls } \widetilde{x} \geq 0 \\ \texttt{lnp1}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon_0}{1+\varepsilon_0}\right) < \texttt{lnp1}(\widetilde{x}) \odot (Z1m \oslash N1m) & \text{falls } \widetilde{x} < 0, \end{cases}$$

dabei lassen sich zu gegebenem $\varepsilon_0 = 2.5082 \cdot 10^{-16}$ die ganzzahligen Werte

$$Z1p = 4503599627370501.0 \qquad N1p = 4503599627370496.0$$
$$Z1m = 9007199254740986.0 \qquad N1m = 9007199254740992.0$$

mit Hilfe der Funktion `eps2fractions()` berechnen, vgl. dazu als Beispiel das Programm von Seite 80. Man kann nun den geeigneten Bereich für die Oberschranke $\widetilde{x}$ dadurch bestimmen, dass man für positive und negative $\widetilde{x}$-Werte experimentell die Stellen sucht, an denen die beiden möglichen Oberschranken etwa den gleichen Wert haben. Man findet die beiden Stellen $2 \cdot 10^{-15}$ und -10^{-15}:

$$\ln(1+\widetilde{x}) \leq \begin{cases} \widetilde{x} & \text{falls } 0 \leq \widetilde{x} \leq 2 \cdot 10^{-15} \\ \texttt{lnp1}(\widetilde{x}) \odot (Z1p \oslash N1p) & \text{falls } \widetilde{x} > 2 \cdot 10^{-15} \end{cases}$$

$$\ln(1+\widetilde{x}) \le \begin{cases} \widetilde{x} & \text{falls } -10^{-15} \le \widetilde{x} < 0 \\ \texttt{lnp1}(\widetilde{x}) \odot (Z1m \oslash N1m) & \text{falls } -1 < \widetilde{x} < -10^{-15} \end{cases}$$

1.3 Speicherung von IEEE-Konstanten

Bei der Implementierung von Standardfunktionen nach dem auf P.T.P. Tang [27] zurückgehenden Tabellenverfahren steht man vor der scheinbar einfachen Aufgabe, entweder schon berechnete **IEEE**-Zahlen zu speichern oder vorgegebene dezimale Näherungswerte zunächst zur nächsten **IEEE**-Zahl $\widetilde{x} \in S(2,53)$ zu runden und dann den Wert von $\widetilde{x}$ in einem Modul durch eine einfache Rechenvorschrift mit bekannten Operanden genau zu reproduzieren. Die benutzten Operanden müssen natürlich selbst wieder **IEEE**-Zahlen sein, und zur besseren Lesbarkeit sollen diese Operanden in dezimaler Form exakt darstellbar sein.

Die gestellte Aufgabe ist lösbar, da eine beliebige[2], aber betragsmäßig nicht zu kleine Binärzahl $\widetilde{x} \in S(2,53)$ des *double*–Formats als Quotient zweier ganzzahliger *double*–Zahlen z, n exakt darstellbar ist:

$$\widetilde{x} = \frac{z}{n} = z \oslash n, \quad \widetilde{x}, z, n \in S(2,53) \quad \text{und} \quad z, n \in \mathbb{Z},\ n \neq 0; \tag{1.11}$$

Zum Nachweis schreiben wir $\widetilde{x}$ zunächst nach (1.3) von Seite 4 in der Form

$$\widetilde{x} = \pm m \cdot 2^e, \quad \text{mit} \quad m = \sum_{i=0}^{52} d_i \cdot 2^{-i} = d_0 \,.\, d_1 d_2 \ldots d_{52}$$

Im Falle $e \ge 52$ ist dann $\widetilde{x}$ selbst schon ganzzahlig, und wir wählen $z = \widetilde{x}$ und $n = 1$. Für $e < 52$ berechnen wir den Zähler z als Produkt

$$z = \widetilde{x} \cdot 2^{ex}$$

und verlangen $e + ex - 52 = 0$. Dadurch wird dann $z \in S(2,53)$ ganzzahlig, und mit $n := 2^{ex} = 2^{52-e} \in \mathbb{Z}$ folgt schließlich die Behauptung.

Die konkrete Berechnung der ganzzahligen Werte $z, n \in \mathbb{Z}$ zu einem vorgegebenen $\widetilde{x} \in S(2,53)$ erfolgt mit Hilfe der Funktion

```
void IEEE2frac_int(const real& x, real& z, real& n);
```

die vom Modul `bnd_util` exportiert wird. Beachten Sie bitte, dass die in (1.11) angegebene Quotientendarstellung z/n keinesfalls eindeutig ist. So liefert die Funktion `IEEE2frac_int()` z.B. für $\widetilde{x} = 1$ die Werte $z = n = 4503599627370496.0$, obwohl

[2] Bei betragsmäßig zu kleinem $\widetilde{x}$ müsste n so groß werden, dass $n \in S(2,53)$ nicht mehr erfüllt ist. Für die numerische Praxis ist dies jedoch keine wirkliche Einschränkung!

auch $z = n = 1$ eine zulässige Darstellung wäre. Die berechneten großen Werte für z, n werden begründet durch die Annahme, dass in der Mantisse von $\widetilde{x}$ im ungünstigsten Fall das Bit d_{52} von Null verschieden ist und somit die Mantisse den kleinsten Summanden $1 \cdot 2^{-52}$ besitzt, der dann durch Multiplikation mit 2^{ex} ganzzahlig zu machen ist. In unserem speziellen Fall $\widetilde{x} = 1$ gilt natürlich: $d_1 = d_2 = \ldots = d_{52} = 0$. Im nächsten Beispiel soll der dezimale Näherungswert $0.1 \notin S(2, 53)$ zunächst zur nächsten Rasterzahl $\widetilde{x} = \mathtt{x} \in S(2, 53)$ gerundet werden. Dies wird realisiert durch die **C-XSC** Anweisungen:

```
cout << "real x = ? ";   cin >> RndNext >> x;
```

und der Funktionsaufruf `IEEE2frac_int(x,z,n);` liefert dann mit den ganzzahligen Werten `z` = 7205759403792794.0 und `n` = 72057594037927936.0 den gewünschten Quotienten `x` = `z`/`n`. Beachten Sie bitte, dass nach dem **IEEE**–Standard die Beziehung `z`/`n` = `z`⊘`n` gilt, da der exakte Quotient `z`/`n` $\in S(2, 53)$ nach Voraussetzung eine Maschinenzahl ist. Bei der konkreten Maschinendivision `z` ⊘ `n` braucht man sich also um einen eventuell voreingestellten Rundungsmodus nicht zu kümmern! Beachten Sie bitte weiter, dass eine anschließende dezimale Ausgabe von `x` mit Hilfe der Anweisung

```
cout << "x = " << SetPrecision(23,23) << x << endl;
```

nicht 0.1 liefert, sondern die dezimale Näherung 0.10000000000000000555112 für die von 0.1 verschiedene Maschinenzahl `x` = 0.10000000000000000055511 ... ≈ 0.1. Das folgende Programm `book_expl01.cpp`

```
#include "bnd_util.hpp" // Wegen IEEE2frac_int(x,z,n);
#include <iostream>     // Wegen cout, cin;
using namespace cxsc;
using namespace std;
int main()
{
    real x,z,n;
    while(1) // Endlosschleife
    {
        cout << "real x = ? ";  cin >> RndNext >> x;
        IEEE2frac_int(x,z,n);
        cout << SetPrecision(1,1) << Fixed << "z = " << z << endl;
        cout << "n = " << n << endl;
    }
}
```

liefert zu vorgegebenen dezimalen Näherungen die in nachfolgender Tabelle zusammengestellten ganzzahligen Werte für $\mathtt{z}, \mathtt{n} \in S(2,53)$ in dezimaler Darstellung.

Beachten Sie bitte, dass in der obigen `while(1)`-Schleife mit der Anweisung

```
cout << SetPrecision(1,1) << Fixed << "z = " << z << endl;
```

und dem Manipulator `Fixed` erreicht wird, dass die ganzzahligen Werte $\mathtt{z}, \mathtt{n}$ mit der gewählten Festkommadarstellung auch beim Auftreten vieler Dezimalziffern stets vollständig und damit korrekt ausgegeben werden.

Dezimale Näherung	$z \in S(2,53)$	$n \in S(2,53)$
1.0	4503599627370496.0	4503599627370496.0
-0.5	-4503599627370496.0	9007199254740992.0
0.1	7205759403792794.0	72057594037927936.0
0.1666666666666667	6004799503160663.0	36028797018963968.0
$5.551115123125783 \cdot 10^{-17}$	4503599627370496.0	811296384146066816957890051440 64.0

Tabelle 1.1: Ganzzahlige Werte $z, n \in S(2,53)$, mit $x = z/n \in S(2,53)$

1.4 Absolute und relative Fehler

Ist $a \in \mathbb{R}$, mit $a \neq 0$ das **exakte** Ergebnis einer Rechnung, und bezeichnet man mit $\widetilde{a}$ das i.a. fehlerbehaftete Maschinenergebnis, so ist der **relative Fehler** ε definiert durch:

$$\varepsilon = \frac{\widetilde{a} - a}{a} \quad \Longleftrightarrow \quad \widetilde{a} = a \cdot (1 + \varepsilon)$$

Wenn im Fall $a = 0$ auch der Computerwert $\widetilde{a}$ verschwindet, was in allen praktischen Fällen eintreten wird, so bleibt die obige zweite Gleichung gültig, und der unbestimmte relative Fehler ε wird auf Null gesetzt.
Wenn jedoch im Fall $a = 0$ der Computerwert $\widetilde{a}$ von Null verschieden ist, so kann ein relativer Fehler nicht mehr angegeben werden. Wir betrachten dazu das folgende **Beispiel:**

> Rundet man 1.234567890123458 in **C-XSC** zur nächsten Rasterzahl $\widetilde{r} \in \mathbf{S}(2,53)$ und bestimmt $a = 5 \cdot \widetilde{r} - 4 \cdot \widetilde{r} - \widetilde{r} \equiv 0$ auf dem Rechner, so erhält man das Maschinenergebnis:
>
> $$\widetilde{a} := 5 \odot \widetilde{r} \ominus 4 \odot \widetilde{r} \ominus \widetilde{r} = -2.220446 \ldots \cdot 10^{-16} \neq a = 0\,.$$
>
> In diesem Fall kann nur der absolute Fehler angegeben werden!

Der **absolute Fehler** Δ ist definiert durch: $\Delta = \widetilde{a} - a$, und mit dem relativen Fehler besteht der Zusammenhang:

$$\Delta = \varepsilon \cdot a \tag{1.12}$$

Mit Hilfe des relativen Fehlers ε kann also der absolute Fehler Δ immer berechnet werden, während der relative Fehler ε mit Hilfe des absoluten Fehlers Δ nur bestimmt werden kann, wenn gilt: $a \neq 0$.
Da man in der Praxis zu einem vorgegebenen arithmetischen Ausdruck den absoluten Fehler oft einfacher bestimmen kann als den relativen Fehler, wird man zunächst meist Δ berechnen und danach den relativen Fehler ε, indem man Δ durch das Minimum des Betrags des arithmetischen Ausdrucks dividiert. Beachten Sie bitte, dass alle Überlegungen dieses Abschnitts auch dann richtig bleiben, wenn Zwischenergebnisse oder der Wert des arithmetischen Ausdrucks selbst im denormalisierten Zahlenbereich liegt. Die Praxis zeigt ferner, dass es meist ausreicht, nur den absoluten Fehler zu berechnen, wenn der arithmetischen Ausdruck im denormalisierten Zahlenbereich liegt. Der denormalisierte Zahlenbereich wird oft auch als Unterlaufbereich bezeichnet und mit U = $(-$`MinReal`$, +$`MinReal`$)$ symbolisiert.

1.5 Maximal genaue und hochgenaue Arithmetik

Soll im System **S**(2,53,-1022,+1023) $\equiv$ **S**(2,53) mit den beiden Summanden

$$\widetilde{a} = 1 \in \mathbf{S}(2,53) \quad \text{und} \quad \widetilde{b} = 2^{-53} \in \mathbf{S}(2,53)$$

die Summe $\widetilde{a} + \widetilde{b} = 1 + 2^{-53}$ auf der Maschine näherungsweise berechnet werden, so gilt, wenn $\oplus$ die Maschinenaddition bedeutet:

$$\widetilde{a} \oplus \widetilde{b} = 1 \neq \widetilde{a} + \widetilde{b} = 1 + 2^{-53} = 1 + 1.11022302462515654... \cdot 10^{-16}$$

Das Beispiel zeigt, dass das Maschinenergebnis mit dem exakten Ergebnis i.a. nicht übereinstimmt und dass man daher die vier Maschinenoperatoren $\oplus, \ominus, \odot, \oslash$ mit Operanden aus **S**(2,53) von den entsprechenden Operatoren $+, -, \cdot, /$ mit Operanden aus $\mathbb{R}$ zu unterscheiden hat.
Für den absoluten Fehler Δ_+ gilt: $\Delta_+ = (\widetilde{a} \oplus \widetilde{b}) - (\widetilde{a} + \widetilde{b})$;
Im Fall $(\widetilde{a} + \widetilde{b}) \neq 0$ ist der relative Fehler ε_+ definiert durch:

$$\varepsilon_+ = \frac{(\widetilde{a} \oplus \widetilde{b}) - (\widetilde{a} + \widetilde{b})}{(\widetilde{a} + \widetilde{b})} \quad \text{bzw.} \quad (\widetilde{a} \oplus \widetilde{b}) = (\widetilde{a} + \widetilde{b}) \cdot (1 + \varepsilon_+)\,;$$

Die obige zweite Gleichung gilt zunächst für alle $\widetilde{a}, \widetilde{b} \in \mathbf{S}(2,53)$, wenn $\widetilde{a} + \widetilde{b} \neq 0$. Im Fall $\widetilde{a} + \widetilde{b} = 0$ verlangt eine, durch den **IEEE-Standard** definierte, maximal

genaue Arithmetik: $\widetilde{a} \oplus \widetilde{b} = 0\,,$ d.h. die obige zweite Gleichung gilt dann für alle $\widetilde{a}, \widetilde{b} \in \mathbf{S}(2{,}53)$, und die unbestimmte, relative Fehlergröße ε_+ wird auf Null gesetzt. In ganz entsprechender Weise kann der relative Fehler bei maximal genauer Arithmetik auch für die drei restlichen Operatoren formuliert werden:

$$(\widetilde{a} \bullet \widetilde{b}) = (\widetilde{a} \circ \widetilde{b}) \cdot (1 + \varepsilon)$$

$$\bullet \in \{\oplus, \ominus, \odot, \oslash\}\,; \quad \circ \in \{+, -, \cdot\,, /\}\,; \quad \widetilde{a}, \widetilde{b} \in \mathbf{S}(2{,}53)\,.$$

Für beliebige Rasterzahlen $\widetilde{a}, \widetilde{b} \in \mathbf{S}(2{,}53)$ gilt also i.a.: $\widetilde{a} \bullet \widetilde{b} \neq \widetilde{a} \circ \widetilde{b}\,.$
Bei **maximal genauer Arithmetik** wird weiter verlangt, dass - von Underflow und Overflow abgesehen - das Maschinenergebnis $\widetilde{a} \bullet \widetilde{b}$ durch Rundung von $\widetilde{a} \circ \widetilde{b} \in I\!R$ zur **nächsten Rasterzahl** bestimmt wird. Liegt dann $\widetilde{a} \circ \widetilde{b} \in I\!R$ zwischen den benachbarten normalisierten Rasterzahlen $\widetilde{x}, \widetilde{y}$, so gilt:

$$|\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b}| \leq \frac{1}{2} \cdot (\widetilde{y} - \widetilde{x}),$$

und im Fall $\widetilde{a} \circ \widetilde{b} > 0$ erhält man nach Seite 5 direkt:

$$\left| \frac{\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b}}{\widetilde{a} \circ \widetilde{b}} \right| \leq \frac{1}{2} \cdot \frac{\widetilde{y} - \widetilde{x}}{\widetilde{x}} \leq \frac{1}{2} \cdot 2^{-52} = 2^{-53} =: \varepsilon(m) \tag{1.13}$$

Mit der Schreibweise $\widetilde{a} \bullet \widetilde{b} = \widetilde{a} \circ \widetilde{b} \cdot (1 + \varepsilon)$ gilt dann im normalisierten Bereich bei maximal genauer Arithmetik für ε die Abschätzung:

$$|\varepsilon| \leq \tfrac{1}{2} \cdot 2^{1-53} = 2^{-53} =: \varepsilon(m)\,; \quad \text{maximal genaue Arithmetik}$$

Der absolute Fehler $(\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b})$ wird dann für Ergebnisse im normalisierten Bereich abgeschätzt durch:

$$|\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b}| \leq \varepsilon(m) \cdot |\widetilde{a} \circ \widetilde{b}| \tag{1.14}$$

Im denormalisierten Bereich, d.h. wenn die Ergebnisse der Operationen im Underflowbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$ liegen, gilt bei maximal genauer Arithmetik für den absoluten Fehler mit $\texttt{minreal} = 2^{-1074}$:

$$|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \leq \left.\begin{cases} 0 & \text{für } \circ \in \{+, -\} \\ \texttt{minreal} & \text{für } \circ \in \{\cdot,\ /\ \} \end{cases}\right\} =: \Delta_{rnd,U}(\circ) \tag{1.15}$$

Zum Beweis siehe [3]. Addition und Subtraktion werden also bei maximal genauer Rechnung stets rundungsfehlerfrei ausgeführt, wenn Summe oder Differenz im Unterlaufbereich U liegt. Bei der Multiplikation und Division besitzt die Oberschranke des

absoluten Fehlers den konstanten Wert $\texttt{minreal} = 2^{-1074}$, so dass der entsprechende relative Fehler im Unterlaufbereich U umso größer wird, je kleiner z.B. der Betrag des Quotienten ausfällt.

Bei einigen Rechnersystemen kann man nicht ausschließen, dass Teile eines Algorithmus in einer nur **hochgenauen** Arithmetik ausgewertet werden, um sehr zeitaufwendige Modus–Umschaltungen im Prozessor zu vermeiden. Es ist daher sinnvoll, für den vollständigen Algorithmus eine Fehlerabschätzung unter Berücksichtigung einer nur hochgenauen Rechnerarithmetik durchzuführen. In diesem Fall gilt dann für den Betrag des relativen Fehlers $|\varepsilon|$ die Abschätzung:

$$|\varepsilon| \leq 2 \cdot \varepsilon(m) = 2^{-52} =: \varepsilon(h)\,; \quad \text{hochgenaue Arithmetik}$$

Dabei wird vorausgesetzt, dass die Ergebnisse der vier Grundoperationen nur im normalisierten Zahlenbereich liegen. Fallen diese Ergebnisse jedoch in den denormalisierten Bereich, so gilt mit $\texttt{MinReal} = 2^{-1022}$ für den absoluten Fehler $(\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b})$ bei nur hochgenauer Arithmetik

$$|\widetilde{a} \bullet \widetilde{b} - \widetilde{a} \circ \widetilde{b}| \leq \texttt{MinReal} =: \Delta_{rnd,U}(\circ). \tag{1.16}$$

Bei nur hochgenauer Arithmetik unterscheidet sich das Rechnerergebnis $\widetilde{a} \bullet \widetilde{b}$ vom exakten Wert $\widetilde{a} \circ \widetilde{b}$ höchstens um eine Einheit (1 ulp) an der letzten Mantissenstelle, d.h. zwischen $\widetilde{a} \bullet \widetilde{b}$ und $\widetilde{a} \circ \widetilde{b}$ liegt keine weitere Maschinenzahl. Leider ist der Sprachgebrauch bezüglich maximal genau und hochgenau nicht immer eindeutig; so wird im Gegensatz zu diesem Buch die hochgenaue Arithmetik oft als maximal genau bezeichnet. Will man z.B. die Standardfunktionen für das **IEEE** double-Format auf möglichst vielen Rechnern implementieren, so sollten sich die Fehlerabschätzungen auf den ungünstigsten Fall der nur hochgenauen Arithmetik beziehen, damit die berechneten Fehlerschranken auch noch in diesem Fall garantiert werden können.

Beachten Sie bitte, dass im Gegensatz zur Addition und Subtraktion die relativen Fehlerschranken $\varepsilon(m)$, $\varepsilon(h)$ bei der Multiplikation und Division sehr deutlich überschritten werden, wenn Produkt oder Quotient in den Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$ fallen. Beim Entwurf eines Algorithmus mit minimaler Fehlerschranke sollte man daher unbedingt vermeiden, dass Zwischenergebnisse in U liegen. Es ist aber zu beachten, dass alle in diesem Buch bereitgestellten Werkzeuge zur automatischen Fehlerabschätzung die Fehlerschranken auch dann noch korrekt berechnen, wenn Zwischenergebnisse in den denormalisieten Bereich U fallen!

Kapitel 2

Fehlerschranken für $\{\oplus, \ominus, \odot, \oslash\}$

2.1 Ein Beispiel

In Kapitel 1 haben wir für die vier Grundoperationen $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ die absoluten und relativen Fehlerschranken beschrieben, wenn diese Operationen in maximal genauer oder hochgenauer Arithmetik auf der Maschine durchgeführt werden. Bei all diesen Überlegungen war jedoch vorausgesetzt, dass die Operanden $\widetilde{a}$, $\widetilde{b}$ dabei stets Maschinenzahlen des double–Systems $\mathbf{S}(2, 53, -1022, +1023) \equiv \mathbf{S}(2, 53)$ sind. Wenn jetzt aber auf einer solchen Maschine z.B. die Summe $\pi + \sqrt{2}$ näherungsweise zu berechnen ist, so ergeben sich für die Abschätzung des dabei aufgetretenen absoluten Fehlers die beiden folgenden Probleme:

1. Wie erhält man auf der Maschine brauchbare Näherungen $\widetilde{p}$ und $\widetilde{w}$ für die in $\mathbf{S}(2, 53)$ nicht darstellbaren reellen Zahlen π und $\sqrt{2}$ und welche absoluten Fehler Δ_π bzw. $\Delta_{\sqrt{2}}$ sind damit verbunden?

2. Wie berechnet man aus den Fehlern $\Delta_\pi, \Delta_{\sqrt{2}}$ und aus dem absoluten Fehler der Maschinenaddition $\widetilde{p} \oplus \widetilde{w}$ den absoluten Gesamtfehler Δ?

Wir nehmen zunächst an, dass wir Verfahren kennen, mit denen man π und $\sqrt{2}$ maximal genau in die Rasterzahlen $\widetilde{p}$ und $\widetilde{w}$ runden kann. Es gilt dann nach (1.12) auf Seite 14 für $\widetilde{p}$ und $\widetilde{w}$:

$$\begin{aligned} \widetilde{p} &= \pi + \Delta_\pi \quad \text{mit} \quad |\Delta_\pi| \leq \varepsilon(m) \cdot \pi = \Delta(\pi) \\ \widetilde{w} &= \sqrt{2} + \Delta_{\sqrt{2}} \quad \text{mit} \quad |\Delta_{\sqrt{2}}| \leq \varepsilon(m) \cdot \sqrt{2} = \Delta(\sqrt{2}) \end{aligned}$$

Der bei der Addition fortgepflanzte Datenfehler $(\pi + \sqrt{2}) - (\widetilde{p} + \widetilde{w})$ lässt sich dann mit Hilfe der Dreiecksungleichung wie folgt abschätzen:

$$\begin{aligned} |(\pi + \sqrt{2}) - (\widetilde{p} + \widetilde{w})| &= |\pi + \sqrt{2} - (\pi + \Delta_\pi + \sqrt{2} + \Delta_{\sqrt{2}})| \\ &= |\Delta_\pi + \Delta_{\sqrt{2}}| \\ &\leq \Delta(\pi) + \Delta(\sqrt{2}) =: \Delta_{dat}(+) \end{aligned}$$

Für die Maschinenaddition gilt ebenfalls nach (1.12) von Seite 14:

$$\widetilde{p} \oplus \widetilde{w} = (\widetilde{p} + \widetilde{w}) + \Delta_+ \quad \text{mit} \quad |\Delta_+| \leq \varepsilon(m) \cdot |\widetilde{p} + \widetilde{w}|$$

und für den Gesamtfehler $(\pi + \sqrt{2}) - (\widetilde{p} \oplus \widetilde{w})$ findet man schließlich die Abschätzung:

$$\begin{aligned} |(\pi + \sqrt{2}) - (\widetilde{p} \oplus \widetilde{w})| &= |\pi + \sqrt{2} - (\widetilde{p} + \widetilde{w} + \Delta_+)| \\ &= |\pi + \sqrt{2} - (\pi + \Delta_\pi + \sqrt{2} + \Delta_{\sqrt{2}} + \Delta_+)| \\ &= |\Delta_\pi + \Delta_{\sqrt{2}} + \Delta_+| \leq \Delta_{dat}(+) + |\Delta_+| \\ &\leq \Delta_{dat}(+) + \varepsilon(m) \cdot |\widetilde{p} + \widetilde{w}| \\ &= \Delta_{dat}(+) + \varepsilon(m) \cdot |\pi + \Delta_\pi + \sqrt{2} + \Delta_{\sqrt{2}}| \\ &\leq \Delta_{dat}(+) + \varepsilon(m) \cdot \Big\{(\pi + \sqrt{2}) + \Delta(\pi) + \Delta(\sqrt{2})\Big\} \\ &= \Delta_{dat}(+) + \varepsilon(m) \cdot \Big\{(\pi + \sqrt{2}) + \Delta_{dat}(+)\Big\} \end{aligned}$$

Mit den Bezeichnungen

$$\Delta_{dat}(+) := \Delta(\pi) + \Delta(\sqrt{2}) = \varepsilon(m) \cdot (\pi + \sqrt{2}) \tag{2.1}$$

$$\Delta_{rnd}(+) := \varepsilon(m) \cdot \Big\{(\pi + \sqrt{2}) + \Delta_{dat}(+)\Big\}, \quad \varepsilon(m) := 2^{-53} \tag{2.2}$$

gilt dann für die Abschätzung des Gesamtfehlers bei maximal genauer Arithmetik:

$$|(\pi + \sqrt{2}) - (\widetilde{p} \oplus \widetilde{w})| \leq \Delta_{dat}(+) + \Delta_{rnd}(+) \tag{2.3}$$

Mit $\Delta_{dat} = 2^{-53} \cdot (\pi + \sqrt{2}) < 5.058 \cdot 10^{-16}$ und $\Delta_{rnd}(+) < 2^{-53} \cdot (\pi + \sqrt{2} + + 5.058 \cdot 10^{-16}) < 5.058 \cdot 10^{-16}$ folgt schließlich:

$$|(\pi + \sqrt{2}) - (\widetilde{p} \oplus \widetilde{w})| < 1.012 \cdot 10^{-15} =: S_1 \tag{2.4}$$

S_1 ist damit der größtmögliche Wert (worst case error) für den absoluten Fehler, wenn man $\pi + \sqrt{2}$ auf dem Rechner maximal genau auswertet. Um den real auftretenden absoluten Fehler zu bestimmen, kann man mit dem folgenden Programm `book_expl02.cpp` zunächst den Maschinenwert $(\widetilde{p} \oplus \widetilde{w})$ berechnen:

```
//*************************************************************************
   #include <iostream>  // Wegen cout
   #include <real.hpp>  // Wegen using namespace cxsc;
   using namespace cxsc;
   using namespace std;

   int main()
   {
       real p,w;
       string pi = "3.141592653589793";
       string w2 = "1.41421356237309504";
       pi >> RndNext >> p; // pi und w2 werden maximalgenau
       w2 >> RndNext >> w; // nach p bzw. w gerundet.
       cout << SetPrecision(30,30) << "Maschinensumme p+w = "
            << p+w << endl;
   }
//*************************************************************************
```

Das obige Programm `book_expl02.cpp` liefert die Bildschirmausgabe:

```
Maschinensumme p+w = 4.555806215962888039427980402252
```

Dabei bedeutet $\texttt{p} + \texttt{w} = \widetilde{p} \oplus \widetilde{w}$ die auf der Maschine berechnete Summe, wobei $\widetilde{p}$ bzw. $\widetilde{w}$ die zu π bzw. $\sqrt{2}$ nächstgelegenen Rasterzahlen sind. Vergleicht man mit dem exakten Wert

$$\pi + \sqrt{2} = 4.55580621596288882872643321074\ldots,$$

so erhält man für den tatsächlichen absoluten Fehler die Differenz

$$(\pi + \sqrt{2}) - (\widetilde{p} \oplus \widetilde{w}) = 2.47836\ldots \cdot 10^{-16}.$$

Der tatsächliche absolute Fehler ist damit etwa um den Faktor 4 kleiner als seine berechnete Oberschranke $S_1 = 1.012 \cdot 10^{-15}$.

Anmerkungen:

- Der absolute Fehler wird betragsmäßig am größten, wenn sowohl π und $\sqrt{2}$ als auch $\widetilde{p} + \widetilde{w}$ genau zwischen jeweils zwei benachbarten Rasterzahlen liegen.
- Es gilt: $\pi + \sqrt{2} \in [(\widetilde{p} \oplus \widetilde{w}) - S_1, (\widetilde{p} \oplus \widetilde{w}) + S_1]$. Es wäre jedoch sehr aufwendig, eine Einschließung von $\pi + \sqrt{2}$ nach dieser Methode zu gewinnen. Mit Hilfe der in **C-XSC** existierenden Intervall-Standardfunktionen kann man solche Einschließungen sehr viel einfacher berechnen. Aber gerade zur Realisierung dieser Standardfunktionen sind die in diesem Buch beschriebenen Fehlerabschätzungen zwingend notwendig!

Mit den folgenden Programmbeispielen wollen wir jetzt noch zeigen, wie einfach man mit Hilfe der in **C-XSC** existierenden Intervallarithmetik und den implementierten Intervallstandardfunktionen eine Einschließung für $\pi + \sqrt{2}$ berechnen kann. Eine Einschließung für π erhält man mit der Beziehung $\pi = 4 \cdot \arctan(1)$, wobei das Argument 1 durch den Konstruktor-Aufruf `interval(1)` als Intervallargument zu benutzen ist. Entsprechend ist das Argument 2 durch das Punktintervall `interval(2)` zu ersetzen, um eine Einschließung für $\sqrt{2}$ zu berechnen.

```
#include <iostream>      // Für cout
#include <imath.hpp>     // Für die Intervallstandardfunktionen

using namespace std;
using namespace cxsc;

int main()
{
    interval x;
    x = 4*atan(interval(1)) + sqrt(interval(2));
    cout << SetPrecision(15,15)
         << "pi + sqrt(2) enthalten in: " << x << endl;
}
```

Das obige Programm `book_expl03.cpp` liefert die folgende Bildschirmausgabe:

```
pi + sqrt(2) enthalten in: [4.555806215962881,4.555806215962897]
```

Anmerkungen:

- Der Aufruf `atan(interval(1))` liefert eine Einschließung für $\arctan(1)$
- `4*atan(interval(1))` liefert eine Einschließung für $\pi = 4 \cdot \arctan(1)$
- Der Aufruf `sqrt(interval(2))` liefert eine Einschließung für $\sqrt{2}$
- Die Anweisung `x = 4*atan(interval(1)) + sqrt(interval(2));` liefert die gesuchte Einschließung von $\pi + \sqrt{2}$, dabei bedeutet in der obigen Anweisung das Zeichen `+` den Additionsoperator für die beiden Intervalloperanden.

Falls die obige Einschließung für $\pi + \sqrt{2}$ mit ihren 14 korrekten Dezimalstellen nicht genau genug ist, kann man mit Hilfe einer in **C-XSC** implementierten staggered Arithmetik diese Einschließung noch weiter verbessern. Wählt man dazu im nachfolgenden Programm z.B. `stagprec = 2`, so wird die gewünschte Einschließung mit einer internen Präzision von etwa $2 \cdot 16 = 32$ Dezimalziffern durchgeführt.

```
#include <iostream>        // Für cout
#include <l_interval.hpp>  // Für die staggered Arithmetik

using namespace std;
using namespace cxsc;

int main()
{
    stagprec = 2;
    l_interval x;
    x = 4*atan(l_interval(1)) + sqrt(l_interval(2));
    cout << SetDotPrecision(16*stagprec,16*stagprec)
         << "pi + sqrt(2) enthalten in: " << endl;
    cout    << x << endl;
}
```

Das obige Programm `book_expl04.cpp` liefert die folgende Bildschirmausgabe:

```
pi + sqrt(2) enthalten in:
[4.555806215962888287264332107489 10,4.555806215962888287264332107489 30]
```

Beachten Sie bitte, dass die von den beiden Programmen ausgegebenen Intervalle die Summe $\pi + \sqrt{2}$ bewiesenermaßen einschließen. Um dies jedoch mathematisch garantieren zu können, müssen die benutzten Intervallstandardfunktionen die beiden Summanden $\pi = 4 \cdot \arctan(1)$ und $\sqrt{2}$ ebenfalls garantiert einschließen. Zur Realisierung der Intervallstandardfunktionen benötigt man daher garantierte Fehlerschranken der entsprechenden Standardfunktionen mit punktförmigen Maschinenargumenten. Zur Berechnung dieser Fehlerschranken müssen dann u.a. die Fehler abgeschätzt werden, die bei Durchführung der vier Grundoperationen auf der Maschine auftreten. Solche Fehler entstehen z.B. bei der Berechnung der Summe $\widetilde{a} + \widetilde{b}$ zweier Maschinenzahlen $\widetilde{a}, \widetilde{b}$ auf der Maschine, wenn der exakte Wert $\widetilde{a} + \widetilde{b}$ im Raster des double-Formats nicht darstellbar ist, und daher etwa zur nächstgelegenen Rasterzahl $\widetilde{a} \oplus \widetilde{b} \neq \widetilde{a} + \widetilde{b}$ gerundet werden muss.

Es ist Aufgabe dieses Buches, den Anwender zunächst mit der Problematik einer für eine zuverlässige Numerik notwendigen Fehlerabschätzung vertraut zu machen. Nach Durchsicht der entsprechenden Kapitel soll der Leser dann in der Lage sein, für eigene Algorithmen, die im double-Format des **IEEE** Systems auf der Maschine ausgeführt werden sollen, garantierte Fehlerschranken eigenständig zu berechnen.

2.2 Intervalloperanden

Im Beispiel des letzten Abschnitts 2.1 haben wir den absoluten Fehler abgeschätzt, wenn auf einem Gleitkommarechner die Summe $\pi + \sqrt{2}$ näherungsweise berechnet wird. Die beiden punktförmigen Operanden π und $\sqrt{2}$ waren dabei im Gleitkommaraster nicht exakt darstellbar.

Wir wollen die Aufgabenstellung jetzt in zwei Schritten verallgemeinern, indem wir zunächst für die vier Grundoperatoren $\{+, -, \cdot, / \}$ als Operanden reelle Intervalle $A, B \in I\!I\!R$ zulassen und diese Intervalle als exakte Wertebereiche zweier reeller Funktionen auffassen. Im zweiten Schritt nehmen wir an, dass z.B. der Wertebereich A auf einem Rechner fehlerhaft ausgewertet wurde. Ist nun $a \in A$ ein exakter Funktionswert aus dem Wertebereich A und ist $\widetilde{a}$ das auf einem Rechner fehlerhaft ausgewertete Maschinenergebnis, so soll für alle $a \in A$ und ganz entsprechend für alle $b \in B$ folgender Zusammenhang bestehen:

$$\widetilde{a} = a + \Delta_a, \text{ mit } |\Delta_a| \leq \Delta(a) \text{ für alle } a \in A \tag{2.5}$$

$$\widetilde{b} = b + \Delta_b, \text{ mit } |\Delta_b| \leq \Delta(b) \text{ für alle } b \in B \tag{2.6}$$

Für die fehlerbehafteten Rasterzahlen $\widetilde{a}$ und $\widetilde{b}$ können noch die folgenden Einschließungen angegeben werden:

$$\widetilde{a} \in \widetilde{A} := A + [-\Delta(a), +\Delta(a)] \text{ für alle } a \in A \tag{2.7}$$

$$\widetilde{b} \in \widetilde{B} := B + [-\Delta(b), +\Delta(b)] \text{ für alle } b \in B \tag{2.8}$$

Die Aufgabenstellung besteht nun darin, z.B. absolute Fehlerschranken $\Delta(\circ)$ so zu berechnen, dass für $\circ \in \{+, -, *, / \}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ gilt:

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta(\circ) \quad \text{für alle } a \in A \text{ und für alle } b \in B. \tag{2.9}$$

Unter der Voraussetzung $a \circ b \neq 0$ verlangt man für die relativen Fehlerschranken $\varepsilon(\circ)$ entsprechend:

$$\left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon(\circ) \quad \text{für alle } a \in A \text{ und für alle } b \in B. \tag{2.10}$$

Bedeutet $\langle A \rangle$ das Minimum aller Betragselemente aus A, so besteht im Falle $0 \notin A$ zwischen den absoluten und relativen Fehlern Δ_a und ε_a bzgl. aller $a \in A$ noch folgender Zusammenhang:

$$\varepsilon_a = \frac{\widetilde{a} - a}{a} \equiv \frac{\Delta_a}{a} \quad \longrightarrow \quad |\varepsilon_a| \leq \frac{\Delta(a)}{|a|} \leq \frac{\Delta(a)}{\langle A \rangle} =: \varepsilon(a) \tag{2.11}$$

Für alle $a \in A$ ist damit $\varepsilon(a)$ die Oberschranke aller relativen Fehler ε_a.

Anmerkungen:

- $A = [\underline{a}, \overline{a}] \in I\mathbb{R}$ ist ein reelles Intervall, dessen Grenzen $\underline{a}, \overline{a} \in \mathbb{R}$ i.a. keine Elemente des double–Systems $\mathbf{S}(2, 53)$ sind. Bei der praktischen Berechnung der Fehlerschranken $\Delta(\circ)$ oder $\varepsilon(\circ)$ muss ein solches Intervall A durch ein Maschinenintervall `A` der **C-XSC** Umgebung optimal eingeschlossen werden. Für $A = [0.1, 2.01]$ liefert der folgende **C-XSC** Programmausschnitt mit `A` eine solche optimale Einschließung:

```
using namespace cxsc;
interval A;
char* string = "[0.1,2.01]";
string >> A;
```

  Diese notwendigen Einschließungen für $A \subseteq$ `A` verursachen dann bei der Berechnung der Fehlerschranken auf der Maschine i.a. nur leichte Überschätzungen der tatsächlichen Fehlerschranken $\Delta(\circ), \varepsilon(\circ)$. Beachten Sie bitte, dass mit einer Anweisung `A = interval(0.1,2.01)` keine Einschließung des Intervalls $[0.1, 2.01]$ berechnet wird, da der Compiler zunächst die Werte 0.1 und 2.01 zur jeweils *nächsten* Rasterzahl rundet und erst danach der entsprechende Intervall-Konstruktor aufgerufen wird, vgl. auch das Programm auf Seite 31.

- Auch bei der Auswertung von $\widetilde{A} := A + [-\Delta(a), +\Delta(a)]$ muss die Addition auf der Maschine als Intervalladdition realisiert werden, um auch jetzt auf der Maschine eine Einschließung für $\widetilde{A}$ berechnen zu können. Gilt z.B. $\Delta(a) \leq$ `D` $\in \mathbf{S}(2, 53)$, so liefert der folgende Programmteil mit `As` eine optimale Einschließung für $\widetilde{A} \subset$ `As`:

```
using namespace cxsc;
real D;  // D ist Oberschranke von Delta(a)
interval A;
char* string = "[0.1,2.01]";
string >> A;
interval As = A + interval(-D,D);
```

  Beachten Sie bitte, dass auch die absolute Fehlerschranke $\Delta(a) \in \mathbb{R}$ i.a. keine Maschinenzahl ist, und daher im Programm vorher durch `D` $\in \mathbf{S}(2, 53)$ nach oben abgeschätzt werden muss!

2.3 Absolute Fehlerschranken

$|A| \in I\!R$ bzw. $\langle A \rangle \in I\!R$ definieren das Maximum bzw. Minimum aller Betragselemente aus A, und $\circ, \bullet$ symbolisieren die vier Grundoperationen:

$$\circ \in \{+, -, \cdot, / \}, \quad \bullet \in \{\oplus, \ominus, \odot, \oslash\}$$

Für alle $a \in A$ mit $|a - \widetilde{a}| \leq \Delta(a)$ bzw. $|a - \widetilde{a}| \leq \varepsilon(a)|A|$ und für alle $b \in B$ mit $|b - \widetilde{b}| \leq \Delta(b)$ bzw. $|b - \widetilde{b}| \leq \varepsilon(b)|B|$ soll nun eine absolute Fehlerschranke $\Delta(\circ) \in I\!R^+$ angegeben werden, so dass gilt:

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta(\circ) \tag{2.12}$$

Die Abschätzung erfolgt dabei in zwei Schritten:

$$\begin{aligned} |(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| &\leq |(a \circ b) - (\widetilde{a} \circ \widetilde{b})| + |(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \\ &\leq \Delta_{dat}(\circ) + \Delta_{rnd}(\circ) =: \Delta(\circ) \end{aligned} \tag{2.13}$$

Mit $\Delta_{dat}(\circ)$ berechnet man also zuerst eine Oberschranke des fortgepflanzten Datenfehlers $(a \circ b) - (\widetilde{a} \circ \widetilde{b})$ und danach mit $\Delta_{rnd}(\circ)$ eine Oberschranke des Rundungsfehlers $(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})$, der nur durch die Maschinenoperation $(\widetilde{a} \bullet \widetilde{b})$ begründet ist.

2.3.1 Abschätzung des fortgepflanzten Datenfehlers

Die Oberschranken $\Delta_{dat}(\circ)$ des absoluten Datenfehlers

$$|(a \circ b) - (\widetilde{a} \circ \widetilde{b})| \leq \Delta_{dat}(\circ), \quad \circ \in \{+, -, \cdot, / \}$$

sind für alle $a \in A$ und für alle $b \in B$ bei gegebenen absoluten oder relativen Fehlerschranken $\Delta(a), \Delta(b)$ oder $\varepsilon(a), \varepsilon(b)$ in folgender Tabelle zusammengestellt:

gegeben	$\Delta_{dat}(\pm)$	$\Delta_{dat}(\cdot)$	$\Delta_{dat}(/)$
$\Delta(a), \Delta(b)$	$\Delta(a) + \Delta(b)$	$\Delta(a)\Delta(b) + \lvert A\rvert\Delta(b) + \lvert B\rvert\Delta(a)$	$\frac{\Delta(a) + \frac{\lvert A\rvert}{\langle B\rangle}\Delta(b)}{\langle B\rangle - \Delta(b)}$
$\Delta(a), \varepsilon(b)$	$\Delta(a) + \lvert B\rvert\varepsilon(b)$	$\lvert B\rvert\,(\Delta(a)\varepsilon(b) + \lvert A\rvert\varepsilon(b) + \Delta(a))$	$\frac{\Delta(a) + \lvert A\rvert\varepsilon(b)}{\langle B\rangle(1-\varepsilon(b))}$
$\varepsilon(a), \Delta(b)$	$\lvert A\rvert\varepsilon(a) + \Delta(b)$	$\lvert A\rvert\,(\varepsilon(a)\Delta(b) + \Delta(b) + \lvert B\rvert\varepsilon(a))$	$\lvert A\rvert\frac{\varepsilon(a) + \frac{\Delta(b)}{\langle B\rangle}}{\langle B\rangle - \Delta(b)}$
$\varepsilon(a), \varepsilon(b)$	$\lvert A\rvert\varepsilon(a) + \lvert B\rvert\varepsilon(b)$	$\lvert A\rvert \cdot \lvert B\rvert \cdot (\varepsilon(a)\varepsilon(b) + \varepsilon(b) + \varepsilon(a))$	$\frac{\lvert A\rvert(\varepsilon(a) + \varepsilon(b))}{\langle B\rangle(1-\varepsilon(b))}$

Tabelle 2.1: Schranken für die Fortpflanzung des absoluten Datenfehlers

Die Beweise findet man in [3]. Beachten Sie bitte, dass die in Tabelle 2.1 angegebenen absoluten Fehlerschranken $\Delta_{dat}(\circ)$ auf dem Rechner intervallmäßig auszuwerten sind, wenn man garantierte Oberschranken für diese Fehlerschranken erhalten will.

2.3.2 Abschätzung des Rundungsfehlers

Mit $\overline{\varepsilon} \in \{\varepsilon(m), \varepsilon(h)\}$ symbolisieren wir die relative Fehlerschranke der bei der Auswertung eines Terms auf der Maschine jeweils benutzten Arithmetik. Bei maximal genauer Arithmetik gilt $\overline{\varepsilon} = \varepsilon(m) = 2^{-53}$, und bei nur hochgenauer Arithmetik ist $\overline{\varepsilon} = \varepsilon(h) = 2^{-52}$. Der Rundungsfehler ist definiert durch $|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})|$ und ist damit der Betrag der Differenz zwischen dem exakten Ergebnis $(\widetilde{a} \circ \widetilde{b})$ und dem Maschinenergebnis $(\widetilde{a} \bullet \widetilde{b})$, wobei zwei Maschinenzahlen $\widetilde{a}, \widetilde{b}$ durch einen Operator $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ verknüpft werden. Mit den bisherigen Bezeichnungen gilt für die Maschinenzahlen $\widetilde{a}, \widetilde{b}$ nach (2.7) von Seite 22

$$\begin{aligned} \widetilde{a} \in \widetilde{A} &:= A + [-\Delta(a), +\Delta(a)] \\ \widetilde{b} \in \widetilde{B} &:= B + [-\Delta(b), +\Delta(b)] \end{aligned}$$

und für den Rundungsfehler $|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})|$ einer Operation $\circ \in \{+, -, \cdot, /\}$ gilt:

a) im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq U$ und damit auch $\widetilde{A} \circ \widetilde{B} \subseteq U$ bei nur hochgenauer Arithmetik:

$$|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \leq \texttt{MinReal} := \Delta_{rnd,U}(\circ). \tag{2.14}$$

b) Bei Verwendung einer maximal genauen Arithmetik kann die Abschätzung im Unterlaufbereich U noch verschärft werden zu:

$$|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \leq \left.\begin{cases} 0 & \text{für } \circ \in \{+, -\} \\ \texttt{minreal} & \text{für } \circ \in \{\cdot, /\} \end{cases}\right\} := \Delta_{rnd,U}(\circ) \tag{2.15}$$

c) im normalisierten Bereich, d.h. $(\widetilde{A} \bullet \widetilde{B}) \cap U = \emptyset$ und damit $(\widetilde{A} \circ \widetilde{B}) \cap U = \emptyset$

$$|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \leq \overline{\varepsilon} \cdot (\Delta_{dat}(\circ) + |A \circ B|) := \Delta_{rnd,N}(\circ) \tag{2.16}$$

d) im gesamten Bereich, d.h. $(\widetilde{A} \bullet \widetilde{B}) \subseteq [-\texttt{MaxReal}, +\texttt{MaxReal}]$

$$\begin{aligned} |(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| &\leq \max\{\Delta_{rnd,U}(\circ), \Delta_{rnd,N}(\circ)\} \\ &\leq \Delta_{rnd,U}(\circ) + \Delta_{rnd,N}(\circ) := \Delta_{rnd}(\circ) \end{aligned} \tag{2.17}$$

Die Beweise von (2.14) bis (2.17) findet man wieder in [3].

2.3.3 Abschätzung des Gesamtfehlers

Für alle $a \in A$ und für alle $b \in B$ werden bei der Abschätzung des Gesamtfehlers $|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})|$ Oberschranken $\Delta(\circ)$ gesucht für $|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta(\circ)$. Es gilt dabei: $\circ \in \{+, -, \cdot, / \}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$. Nach (2.13) und nach Tabelle 2.1 von Seite 24 erhält man:

a) im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq U$ und damit auch $\widetilde{A} \circ \widetilde{B} \subseteq U$, nach (2.14) bzw. (2.15)

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ) := \Delta(\circ) \tag{2.18}$$

b) im normalisierten Bereich, d.h. $(\widetilde{A} \bullet \widetilde{B}) \cap U = \emptyset$ und damit $(\widetilde{A} \circ \widetilde{B}) \cap U = \emptyset$, nach (2.16)

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta_{dat}(\circ) + \Delta_{rnd,N}(\circ) := \Delta(\circ) \tag{2.19}$$

c) im gesamten Bereich, d.h. $(\widetilde{A} \bullet \widetilde{B}) \subseteq [-\texttt{MaxReal}, +\texttt{MaxReal}]$ und damit auch $(\widetilde{A} \circ \widetilde{B}) \subseteq [-\texttt{MaxReal}, +\texttt{MaxReal}]$, nach (2.17)

$$\begin{aligned} |(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| &\leq \Delta_{dat}(\circ) + \max\{\Delta_{rnd,U}(\circ), \Delta_{rnd,N}(\circ)\} \\ &\leq \Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ) + \Delta_{rnd,N}(\circ) := \Delta(\circ) \end{aligned} \tag{2.20}$$

Auch hier müssen die in (2.18) bis (2.20) angegebenen Summen für $\Delta(\circ)$ auf dem Rechner intervallmäßig ausgewertet werden, um wirklich garantierte Oberschranken des jeweiligen Gesamtfehlers $|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})|$ zu berechnen.

2.4 Relative Fehlerschranken

$|A| \in I\!R$ bzw. $\langle A \rangle \in I\!R$ definieren das Maximum bzw. Minimum aller Betragselemente aus A, und $\circ, \bullet$ symbolisieren die vier Grundoperationen:

$$\circ \in \{+, -, \cdot, / \}, \quad \bullet \in \{\oplus, \ominus, \odot, \oslash\}$$

Für alle $a \in A$ mit $|a - \widetilde{a}| \leq \Delta(a)$ bzw. $|a - \widetilde{a}| \leq \varepsilon(a)|A|$ und für alle $b \in B$ mit $|b - \widetilde{b}| \leq \Delta(b)$ bzw. $|b - \widetilde{b}| \leq \varepsilon(b)|B|$ soll nun eine relative Fehlerschranke $\varepsilon(\circ) \in I\!R^+$ angegeben werden, so dass gilt:

$$\left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon(\circ) \tag{2.21}$$

Die Abschätzung erfolgt wieder in zwei Schritten:

$$\left|\frac{(a\circ b)-(\widetilde{a}\bullet\widetilde{b})}{a\circ b}\right| \le \left|\frac{(a\circ b)-(\widetilde{a}\circ\widetilde{b})}{a\circ b}\right| + \left|\frac{(\widetilde{a}\circ\widetilde{b})-(\widetilde{a}\bullet\widetilde{b})}{a\circ b}\right|$$

$$\le \varepsilon_{dat}(\circ) + \left|\frac{(\widetilde{a}\circ\widetilde{b})-(\widetilde{a}\bullet\widetilde{b})}{a\circ b}\right| \tag{2.22}$$

Mit $\varepsilon_{dat}(\circ)$ berechnet man also zuerst eine Oberschranke des fortgepflanzten relativen Datenfehlers $|\{(a\circ b)-(\widetilde{a}\circ\widetilde{b})\}/(a\circ b)|$. Die Abschätzung des zweiten Summanden rechts in (2.22) erfolgt dann später getrennt für den Unterlaufbereich und für den denormalisierten Zahlenbereich.

2.4.1 Abschätzung des fortgepflanzten Datenfehlers

Die Oberschranken $\varepsilon_{dat}(\circ)$ des fortgepflanzten relativen Datenfehlers

$$\left|\frac{(a\circ b)-(\widetilde{a}\circ\widetilde{b})}{a\circ b}\right| \le \varepsilon_{dat}(\circ), \quad \circ\in\{+,-,\cdot,/\}$$

sind für alle $a\in A=[\underline{a},\overline{a}]$ und für alle $b\in B=[\underline{b},\overline{b}]$ bei gegebenen Fehlerschranken $\Delta(a),\Delta(b)$ oder $\varepsilon(a),\varepsilon(b)$ in folgender Tabelle zusammengestellt:

gegeben	$\varepsilon_{dat}(\pm)$	$\varepsilon_{dat}(\cdot)$	$\varepsilon_{dat}(/)$
$\Delta(a),\Delta(b)$	$\frac{\Delta(a)+\Delta(b)}{\langle A\pm B\rangle}$	$\frac{\Delta(a)}{\langle A\rangle}\cdot\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(a)}{\langle A\rangle}$	$\frac{\frac{\Delta(a)}{\langle A\rangle}+\frac{\Delta(b)}{\langle B\rangle}}{1-\frac{\Delta(b)}{\langle B\rangle}}$
$\Delta(a),\varepsilon(b)$	$\max\limits_{b\in\{\underline{b},\overline{b}\}}\frac{\Delta(a)+\lvert b\rvert\varepsilon(b)}{\langle A\pm b\rangle}$	$\frac{\Delta(a)}{\langle A\rangle}\cdot\varepsilon(b)+\varepsilon(b)+\frac{\Delta(a)}{\langle A\rangle}$	$\frac{\frac{\Delta(a)}{\langle A\rangle}+\varepsilon(b)}{1-\varepsilon(b)}$
$\varepsilon(a),\Delta(b)$	$\max\limits_{a\in\{\underline{a},\overline{a}\}}\frac{\lvert a\rvert\varepsilon(a)+\Delta(b)}{\langle a\pm B\rangle}$	$\varepsilon(a)\cdot\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(b)}{\langle B\rangle}+\varepsilon(a)$	$\frac{\varepsilon(a)+\frac{\Delta(b)}{\langle B\rangle}}{1-\frac{\Delta(b)}{\langle B\rangle}}$
$\varepsilon(a),\varepsilon(b)$	$\max\limits_{a\in\{\underline{a},\overline{a}\},\ b\in\{\underline{b},\overline{b}\}}\frac{\lvert a\rvert\varepsilon(a)+\lvert b\rvert\varepsilon(b)}{\lvert a\pm b\rvert}$	$\varepsilon(a)\varepsilon(b)+\varepsilon(b)+\varepsilon(a)$	$\frac{\varepsilon(a)+\varepsilon(b)}{1-\varepsilon(b)}$

Tabelle 2.2: Schranken für die Fortpflanzung des relativen Datenfehlers

Die Beweise findet man in [3]. Beachten Sie bitte, dass die in Tabelle 2.2 angegebenen relativen Fehlerschranken $\varepsilon_{dat}(\circ)$ auf dem Rechner intervallmäßig auszuwerten sind, wenn man garantierte Oberschranken für diese Fehlerschranken erhalten will. Die in Tabelle 2.2 angegebenen Fehlerschranken $\Delta(a),\varepsilon(a)$ bzw. $\Delta(b),\varepsilon(b)$ beziehen sich auf die vollständigen Intervalle A bzw. B und sind daher bei der Maximumbildung als Konstante anzusehen!

2.4.2 Abschätzung des Rundungsfehlers

Mit $\overline{\varepsilon} \in \{\varepsilon(m), \varepsilon(h)\}$ symbolisieren wir die relative Fehlerschranke der bei der Auswertung eines Terms auf der Maschine jeweils benutzten Arithmetik. Bei maximal genauer Arithmetik gilt $\overline{\varepsilon} = \varepsilon(m) = 2^{-53}$, und bei nur hochgenauer Arithmetik ist $\overline{\varepsilon} = \varepsilon(h) = 2^{-52}$. Nach (2.22) von Seite 27 ist noch eine Oberschranke anzugeben für den Ausdruck $\left|\frac{(\widetilde{a}\circ\widetilde{b})-(\widetilde{a}\bullet\widetilde{b})}{a\circ b}\right|$, der im normalisierten Bereich, d.h. $\widetilde{a} \bullet \widetilde{b} \notin U$ und damit auch $\widetilde{a} \circ \widetilde{b} \notin U$, zunächst durch Erweitern mit $(\widetilde{a} \circ \widetilde{b})$ nach (1.13) wie folgt abgeschätzt werden kann:

$$\begin{aligned}\left|\frac{(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b}\right| &\leq \overline{\varepsilon} \cdot \left|\frac{\widetilde{a} \circ \widetilde{b}}{a \circ b}\right| \\ &\leq \overline{\varepsilon} \cdot \frac{|(a \circ b) - (\widetilde{a} \circ \widetilde{b})| + |a \circ b|}{|a \circ b|} \\ &\leq \overline{\varepsilon} \cdot (\varepsilon_{dat}(\circ) + 1) =: \varepsilon_{rnd,N}(\circ) \qquad (2.23)\end{aligned}$$

Für alle $a \in A$ und für alle $b \in B$ und für die Grundoperationen $\circ \in \{+, -, \cdot, /\}$ erhält man für den Ausdruck $\left|\frac{(\widetilde{a}\circ\widetilde{b})-(\widetilde{a}\bullet\widetilde{b})}{a\circ b}\right|$ in (2.22) mit den Maschinenzahlen $\widetilde{a}, \widetilde{b} \in \mathrm{S}(2, 53)$

a) im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq U$ und damit auch $\widetilde{A} \circ \widetilde{B} \subseteq U$, bei nur hochgenauer Arithmetik:

$$\left|\frac{(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b}\right| \leq \frac{\texttt{MinReal}}{\langle A \circ B\rangle} =: \varepsilon_{rnd,U}(\circ) \qquad (2.24)$$

b) im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq U$ und damit auch $\widetilde{A} \circ \widetilde{B} \subseteq U$, bei maximal genauer Arithmetik:

$$\left|\frac{(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b}\right| \leq \left\{\begin{array}{ll} 0 & \text{für } \circ \in \{+, -\} \\ \frac{\texttt{minreal}}{\langle A \circ B\rangle} & \text{für } \circ \in \{\cdot, /\} \end{array}\right\} =: \varepsilon_{rnd,U}(\circ) \qquad (2.25)$$

c) im normalisierten Bereich, d.h. $\widetilde{A} \bullet \widetilde{B} \cap U = \emptyset$ und damit $\widetilde{A} \circ \widetilde{B} \cap U = \emptyset$

$$\left|\frac{(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b}\right| \leq \overline{\varepsilon} \cdot (\varepsilon_{dat}(\circ) + 1) =: \varepsilon_{rnd,N}(\circ) \qquad (2.26)$$

d) und im gesamten Bereich, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq [-\texttt{MaxReal}, +\texttt{MaxReal}]$

$$\begin{aligned}\left|\frac{(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b}\right| &\leq \max(\varepsilon_{rnd,U}(\circ), \varepsilon_{rnd,N}(\circ)) \\ &\leq \varepsilon_{rnd,U}(\circ) + \varepsilon_{rnd,N}(\circ) =: \varepsilon_{rnd}(\circ) \qquad (2.27)\end{aligned}$$

2.4.3 Abschätzung des Gesamtfehlers

Nach (2.22) und mit den Ergebnissen des letzten Abschnitts kann jetzt der Gesamtfehler abgeschätzt werden. Für alle $a \in A$ und für alle $b \in B$ erhält man jetzt für die vier Grundoperationen $\circ \in \{+, -, \cdot, / \}$ und für die entsprechenden Maschinenoperationen $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$

a) im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq U$ und damit auch $\widetilde{A} \circ \widetilde{B} \subseteq U$

$$\left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon_{dat}(\circ) + \varepsilon_{rnd,U}(\circ) =: \varepsilon(\circ), \tag{2.28}$$

b) im normalisierten Bereich, d.h. $\widetilde{A} \bullet \widetilde{B} \cap U = \emptyset$ und damit $\widetilde{A} \circ \widetilde{B} \cap U = \emptyset$

$$\left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon_{dat}(\circ) + \varepsilon_{rnd,N}(\circ) =: \varepsilon(\circ), \tag{2.29}$$

c) und im gesamten Bereich, d.h. $\widetilde{A} \bullet \widetilde{B} \subseteq [-\texttt{MaxReal}, +\texttt{MaxReal}]$

$$\begin{aligned} \left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| &\leq \varepsilon_{dat}(\circ) + \max(\varepsilon_{rnd,U}(\circ), \varepsilon_{rnd,N}(\circ)) \\ &\leq \varepsilon_{dat}(\circ) + \varepsilon_{rnd,U}(\circ) + \varepsilon_{rnd,N}(\circ) =: \varepsilon(\circ) \end{aligned} \tag{2.30}$$

Auch hier müssen die in (2.28) bis (2.30) angegebenen Summen für $\varepsilon(\circ)$ auf dem Rechner intervallmäßig ausgewertet werden, um wirklich garantierte Oberschranken des jeweiligen relativen Gesamtfehlers zu erhalten. Die Fehlerschranken $\varepsilon_{dat}(\circ)$ des fortgepflanzten relativen Datenfehlers findet man im Abschnitt 2.4.1, und die Schranken $\varepsilon_{rnd,U}(\circ), \varepsilon_{rnd,N}(\circ)$ sind im Abschnitt 2.4.2 bzgl. einer maximalgenauen oder nur hochgenauen Arithmetik für den Unterlaufbereich, für den normalisierten Zahlenbereich oder für den gesamten Zahlenbereich $[-\texttt{MaxReal}, +\texttt{MaxReal}]$ angegeben.

Beachten Sie bitte, dass die Intervalle A, B als die exakten Wertebereiche zweier reeller Funktionen aufzufassen sind und dass ein solcher exakter Funktionswert $a \in A$ auf einem Rechner mit einem fehlerbehafteten Wert $\widetilde{a}$ berechnet wurde. Für alle $a \in A$ sei z.B. folgenden Ungleichungen erfüllt: $|a - \widetilde{a}| \leq \Delta(a)$, dann liegen alle Maschinenzahlen $\widetilde{a}$ im Intervall $\widetilde{A} := A + [-\Delta(a), +\Delta(a)]$.

2.5 Berechnung der Fehlerschranken

In den folgenden Abschnitten werden **C-XSC** Funktionen angegeben, mit denen sich die absoluten und relativen Fehlerschranken berechnen lassen, wenn fehlerbehaftete Operanden mit den vier Maschinenoperationen $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ verknüpft werden. Für eine maximal genaue bzw. nur hochgenaue Arithmetik findet man diese Funktionen in den **C-XSC** Modulen `abs_relm.hpp, abs_relm.cpp` bzw. `abs_relh.hpp, abs_relh.cpp`. Bei hochgenauer Arithmetik sind die vier Sätze 3.1.1 bis 3.1.4 des Abschnitts 3.1 bez. der rundungsfehlerfreien Grundoperationen bei den Funktionen des Moduls `abs_relh` bereits berücksichtigt!

2.5.1 Maximal genaue Arithmetik

Wir wählen wieder die bisherigen Bezeichnungen. A, B sind die Intervalloperanden mit den exakten Operanden $a \in A,\ b \in B$, wobei wir die reellen Intervalle A, B als Wertebereiche reeller Funktionen auffassen können. Wir nehmen jetzt an, dass diese Wertebereiche auf der Maschine fehlerhaft berechnet wurden. An Stelle von $a \in \mathbb{IR}$ wurde also der fehlerbehaftete Maschinenwert $\widetilde{a} \in S(2, 53)$ bestimmt, und für alle $a \in A$ soll gelten: $|a - \widetilde{a}| \leq \Delta(a)$, d.h. die absolute Fehlerschranke $\Delta(a)$ sei bekannt. Falls $0 \notin A$ kann auch vorausgesetzt werden, dass die relative Fehlerschranke $\varepsilon(a)$ gegeben ist, d.h. dass für alle $a \in A$ gilt: $|a - \widetilde{a}| \leq \varepsilon(a) \cdot |a|$. Für die Maschinenzahlen $\widetilde{a}$ gilt: $\widetilde{a} \in \widetilde{A} := A + [-\Delta(a), +\Delta(a)]$. Ganz entsprechende Überlegungen gelten natürlich auch für $B, \widetilde{b}, b, \Delta(b)$ und $\varepsilon(b)$.

Für alle $a \in A$ und für alle $b \in B$ suchen wir jetzt absolute bzw. relative Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$, so dass für $\circ \in \{+, -, \cdot, /\}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ gilt:

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta(\circ) \qquad \text{bzw.} \qquad \left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon(\circ)$$

In den Abschnitten 2.3 und 2.4 wurden für die Fehlerschranken $\Delta(\circ)$ und $\varepsilon(\circ)$ die entsprechenden Formeln zusammengestellt, wobei zu unterscheiden war, ob die Operationsergebnisse $\widetilde{a} \bullet \widetilde{b}$ nur im Unterlaufbereich oder nur im normalisierten Bereich oder in beiden Bereichen liegen. Bei maximal genauer Arithmetik ist in den entsprechenden Formeln der auftretende Wert $\overline{\varepsilon}$ zu ersetzen durch $\overline{\varepsilon} = \varepsilon(m) = 2^{-53}$. Um beispielsweise nach (2.18) für $\Delta(\circ)$ wirklich eine Oberschranke zu erhalten, muss die Summe $\Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ)$ auf der Maschine mit Hilfe der **C-XSC** Funktion `addu` zur nächstgrößeren Maschinenzahl gerundet werden. Ganz entsprechend sind die in den Tabellen 2.1 und 2.2 angegebenen Ausdrücke für $\Delta_{dat}(\circ)$ durch gerichtetes Runden auszuwerten.

Um die garantierten Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$ auf der Maschine berechnen zu können, müssen die reellen Operandenintervalle $A, B \in I\mathbb{IR}$ durch Maschinenintervalle $\mathtt{A}, \mathtt{B} \in IR$ möglichst optimal eingeschlossen werden, d.h. es muss gelten:

$A \subseteq$ `A` und $B \subseteq$ `B`, und diese Maschinenintervalle sind dann die Eingangsparameter der entsprechenden **C-XSC** Funktionen, mit denen die gesuchten Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$ zu berechnen sind. Ganz entsprechend muss man auch für die absoluten und relativen Fehlerschranken $\Delta(a), \Delta(b), \varepsilon(a), \varepsilon(b)$ möglichst kleine Rasterzahlen `dela`, `delb`, `epsa`, `epsb` $\in S(2, 53)$ angeben, so dass gilt:

$$\Delta(a) \leq \texttt{dela},\ \Delta(b) \leq \texttt{delb},\ \varepsilon(a) \leq \texttt{epsa},\ \varepsilon(b) \leq \texttt{epsb}.$$

Jeweils eine dieser Rasterzahlen `dela,delb` und `epsa,epsb` ist dann wieder den entsprechenden **C-XSC** Funktionen als weitere Eingangsparameter zu übergeben.

Rückgabewerte der **C-XSC** Funktionen sind eine Oberschranke für $\Delta(\circ)$ bzw. für $\varepsilon(\circ)$ und ein Maschinenintervall $\texttt{R} \in IR$, das eine Einschließung von $\texttt{A} \circ \texttt{B}$ liefert, d.h. $\texttt{A} \circ \texttt{B} \subseteq \texttt{R} := \texttt{A} \bullet \texttt{B}$; $\circ \in \{+, -, \cdot, /\}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$.

Anhand eines Beispiels soll jetzt gezeigt werden, wie man bei maximal genauer Arithmetik mit Hilfe der im Modul `abs_relm` implementierten **C-XSC** Funktionen eine Fehlerschranke des absoluten Fehlers berechnen kann, wenn der Term $1 + x$ auf der Maschine für alle reellen $x \in B = [1, 2.1]$ ausgewertet wird. Wir interpretieren dabei B wieder als Wertebereich einer reellen Funktion und nehmen an, dass die Funktionswerte $x \in B$ maximal genau berechnet wurden, d.h. es soll für alle $x \in B$ gelten: $|x - \widetilde{x}| \leq \varepsilon(b) \cdot |x|$, mit $\varepsilon(b) = 2^{-53} = 1.11022302\ldots \cdot 10^{-16}$. Es wird also angenommen, dass auf der Maschine die Additionen $1 \oplus \widetilde{x}$ ausgeführt werden. Da der größte absolute Fehler gegeben ist durch $\Delta(x) := \varepsilon(b) \cdot 2.1$, liegen alle Maschinenzahlen $\widetilde{x}$ im Intervall $\widetilde{B} := B + [-\Delta(x), +\Delta(x)]$. Das erste Operandenintervall A ist jetzt ein Punktintervall, d.h. $A = \texttt{A} = [1, 1] = \texttt{interval(1)}$, mit $\Delta(a) = \varepsilon(a) = 0$.

```
#include "abs_relm.hpp" // Wegen abs_addm2(...)
#include <iostream>     // Wegen cout
using namespace cxsc;  using namespace std;
int main()
{
    interval A=interval(1), B,R;   // A = [1,1]
    string("[1,2.1]") >> B; // B schließt [1,2.1] ein!
    real del, dela=0, epsb;
    cin >> RndUp; string("1.11022302E-16") >> epsb;
    abs_addm2(A,dela,B,epsb,R,del); // liefert R und del
    cout << "Absolute Fehlerschranke del = " << del << endl;
    cout << "Relative Fehlerschranke     = "
         << divu(del,Inf(R)) << endl;
    cout << "A+B enthalten in: " << R << endl;
}
```

Das umseitige Programm `book_expl05.cpp` liefert die folgende Bildschirmausgabe:

```
Absolute Fehlerschranke del = 5.773160E-016
Relative Fehlerschranke     = 2.886580E-016
A+B enthalten in: [  2.000000,  3.100001]
```

Anmerkungen:

1. Im umseitigen Programm erreicht man mit: `string("[1,2.1]") >> B;` dass das Intervall $B = [1, 2.1]$ durch das Maschinenintervall $\mathtt{B} \in IR$ optimal eingeschlossen wird. Beachten Sie dabei bitte, dass die Dezimalzahl 2.1 im double-Format nicht darstellbar ist und durch obige Anweisung zur nächstgrößeren Rasterzahl aus $S(2, 53)$ gerundet wird, d.h. es gilt $B = [1, 2.1] \subset \mathtt{B}$.

2. Die Anweisungen: `cin >> RndUp; string("1.11022302E-16") >> epsb;` bewirken, dass die in $S(2, 53)$ nicht darstellbare Zahl $1.11022302 \cdot 10^{-16}$ zur nächstgrößeren Rasterzahl `epsb` gerundet wird, womit garantiert werden kann: $\varepsilon(b) = 2^{-53} <$ `epsb`.

3. Nach obiger Bildschirmausgabe gilt dann für alle $x \in \mathtt{B}$ und damit wegen $B \subset \mathtt{B}$ auch für alle $x \in B$:

$$|(1 + x) - (1 \oplus \widetilde{x})| \leq 5.773160 \cdot 10^{-16}$$

   Dabei wurde für alle exakten $x \in \mathtt{B}$ und damit wegen $B \subset \mathtt{B}$ auch für alle exakten $x \in B$ vorausgesetzt: $|x - \widetilde{x}| \leq \varepsilon(b) \cdot |x|$. Mit $\varepsilon(b) = 2^{-53}$ bedeutet dies, dass die exakten Werte x auf der Maschine nur durch die maximal genau berechneten Maschinenzahlen $\widetilde{x}$ zur Verfügung stehen.

4. Das von der Funktion `abs_addm2(A,dela,B,epsb,R,del)` zurückgegebene Intervall `R` liefert eine garantierte Einschließung der exakten Intervallsumme $[1, 1] + B \subset [1, 1] + \mathtt{B} \subseteq \mathtt{R} := [1, 1] \oplus \mathtt{B}$.

5. Mit Hilfe der berechneten absoluten Fehlerschranke `del` $\leq 5.773160 \cdot 10^{-16}$ kann eine relative Fehlerschranke berechnet werden, indem `del` durch `Inf(R)` dividiert und dabei der exakte Quotient `del`/`Inf`(`R`) mit der **C-XSC** Funktion `divu(del,Inf(R))` zur nächstgrößeren Rasterzahl gerundet wird. Wegen des zu großen Intervalldurchmessers von `R` ist die so berechnete relative Fehlerschranke $2.886580 \cdot 10^{-16}$ eine zu große Überschätzung des tatsächlichen relativen Fehlers. Im nachfolgenden Programmbeispiel werden wir diese kleinere relative Fehlerschranke mit der **C-XSC** Funktion `rel_addm4` aus dem gleichen Modul `abs_relm` berechnen.

Das **C-XSC** Modul `abs_relm` stellt $2 \cdot 4 \cdot 4 = 32$ Funktionen zur Verfügung, um bei einer maximal genauen Arithmetik für die vier Grundoperationen absolute oder relative Fehlerschranken zu berechnen. Der erste Faktor 2 berücksichtigt sowohl die absoluten als auch die relative Fehlerschranken. Der zweite Faktor 4 steht für die vier Grundoperationen, und der letzte Faktor 4 berücksichtigt die vier Zeilen in den Tabellen 2.1 und 2.2 auf den Seiten 24 und 27, wobei in jeder Zeile die Art der jeweiligen Fehlerschranken für die Intervalloperanden A, B festgelegt ist. In der jeweils dritten Zeile wird z.B. festgelegt, dass für den ersten Intervalloperanden A eine relative und für den zweiten Intervalloperanden B eine absolute Fehlerschranke vorgegeben ist.

Die drei ersten Buchstaben des Funktionsnamens `abs` bzw. `rel` bestimmen die Berechnung einer absoluten oder relativen Fehlerschranke. Die drei Buchstaben nach dem Unterstrich bestimmen die gewählte Grundoperation, und der nachfolgende Buchstabe `m` setzt für die vier Grundoperationen eine maximal genaue Arithmetik voraus. Die letzte Ziffer des Funktionsnamens bestimmt schließlich die Tabellenzeile, nach der die Fehlerschranken der Intervalloperanden A, B festgelegt sind. So besagt z.B. der Funktionsname `rel_addm4`, dass bei maximal genauer Arithmetik für die Addition ein relativer Fehler zu berechnen ist, wenn für beide Intervalloperanden A, B jeweils eine relative Fehlerschranke vorgegeben ist. Im folgenden Programmbeispiel wird mit dieser Funktion für $A = [1, 1]$ und $B = [1, 2.1]$ mit $\varepsilon(a) = 0$ und $\varepsilon(b) = 2^{-53}$ der relative Fehler abgeschätzt, wenn die Addition $1 \oplus \widetilde{x}$ in maximal genauer Arithmetik erfolgt:

```
#include "abs_relm.hpp" // Wegen rel_addm4(...)
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;
int main()
{
    interval A=interval(1), B,R;  // A = [1,1]
    string("[1,2.1]") >> B; // B schließt [1,2.1] ein!
    real eps, epsa=0, epsb;
    cin >> RndUp; string("1.11022302E-16") >> epsb;
       // Die Funktion rel_addm4 aus dem Modul abs_relm
    rel_addm4(A,epsa,B,epsb,R,eps);
       // berechnet R und die relative Fehlerschranke eps.
    cout << "Relative Fehlerschranke eps = " << eps << endl;
    cout << "A+B enthalten in R = " << R << endl;
}
```

Das umseitige Programm `book_expl06.cpp` liefert die folgende Bildschirmausgabe:

```
Relative Fehlerschranke eps = 1.862310E-016
A+B enthalten in R = [  2.000000,  3.100001]
```

Anmerkungen:

1. Die Anmerkungen 1,2,4 von Seite 32 können hier direkt übernommen werden.

2. Nach obiger Bildschirmausgabe gilt für alle $x \in$ `B` und damit wegen $B \subset$ `B` auch für alle $x \in B$:

$$\left| \frac{(1+x) - (1 \oplus \widetilde{x})}{1+x} \right| \leq \varepsilon(+) \leq \texttt{eps} = 1.862310 \cdot 10^{-016}$$

   Dabei wurde für alle exakten $x \in$ `B` und damit wegen $B \subset$ `B` auch für alle exakten $x \in B$ vorausgesetzt: $|x - \widetilde{x}| \leq \varepsilon(b) \cdot |x|$. Mit $\varepsilon(b) = 2^{-53}$ bedeutet dies, dass die exakten Werte x auf der Maschine nur durch die maximal genau berechneten Maschinenzahlen $\widetilde{x}$ zur Verfügung stehen.

3. Setzt man $\varepsilon(b) = 0$, so erhält man für den relativen Fehler die Abschätzung $\varepsilon(+) \leq$ `eps` $= 1.110223 \cdot 10^{-016}$, d.h. genau[1] die relative Fehlerschranke $\varepsilon(m) = 2^{-53}$ der Grundoperationen im normalisierten Zahlenbereich. Beachten Sie bitte in diesem Zusammenhang, dass mit $\varepsilon(b) = 0$ das Intervall `B` $= B$ der exakten Operanden $b = \widetilde{b} \in$ `B` nur aus den endlich vielen Maschinenzahlen $\widetilde{b} \in S(2, 53)$ bestehen kann!

4. Im Punkt 5. von Seite 32 hatten wir vermutet, dass der dort mit Hilfe des absoluten Fehlers berechnete relative Fehler $2.886580 \cdot 10^{-16}$ wegen des zu großen Intervalldurchmessers von `R` stark überschätzt sein würde. Diese Vermutung wird jetzt bestätigt, wenn man die genannte Fehlerschranke mit der oben berechneten relativen Fehlerschranke `eps` $= 1.862310 \cdot 10^{-16}$ vergleicht, die durch Aufruf der **C-XSC** Funktion `rel_addm4` im letzten Beispiel berechnet wurde.

Nach Punkt 4. kann jetzt schon die folgende Grundregel formuliert werden:

> Soll für die Auswertung eines Terms $T(x)$ eine relative Fehlerschranke berechnet werden und sind für alle auftretenden Operanden nur relative Fehlerschranken gegeben, so sollte man den relativen Fehler direkt, d.h. also nicht über die absolute Fehlerschranke bestimmen!

[1] `eps` $= 1.110223 \cdot 10^{-016}$ ist natürlich der aufgerundete Wert von $\varepsilon(m) = 2^{-53}$

Die praktische Erfahrung zeigt jedoch, dass bei der Auswertung eines Terms $T(x)$ nicht für alle Operanden eine relative sondern oft nur eine absolute Fehlerschranke angegeben werden kann. In diesem Fall sollte man die folgende zweite Grundregel beachten:

> Soll für die Auswertung eines Terms $T(x)$ eine relative Fehlerschranke berechnet werden und sind für alle auftretenden Operanden nur absolute Fehlerschranken verfügbar, so unterteile man das für die Variable x vorgegebene Intervall $[x]$ in hinreichend viele Teilintervalle $[x_i]$ und berechne dann für jedes $[x_i]$ zunächst den absoluten Fehler und damit nach Division durch `Inf(R)` den relativen Fehler ε_i. Der relative Fehler ε ist dann das Maximum aller berechneten ε_i.

Die oben beschriebene Intervallunterteilung erfordert einen hohen numerischen Aufwand. Dies ist jedoch kein wirklicher Nachteil, da die Fehlerabschätzung für einen Term $T(x)$ nur einmal durchzuführen ist und dies mit den heutigen Rechnern oft in nur wenigen Minuten bewältigt werden kann. Entsprechende Beispiele findet man auf den Seiten 44 und 46.

Abschließend noch eine Bemerkung zur maximal genauen Arithmetik. Führt man auf einem Computer eine der vier Grundoperationen $\{\oplus, \ominus, \odot, \oslash\}$ **ohne** gerichtete Rundung aus, so erhält i.a. maximal genaue Ergebnisse. Dass dies aber nicht immer der Fall sein muss, zeigt schon unter Suse Linux 7.3 die Addition $1 \oplus \widetilde{x}$. Rundet man `0.00011240500094367170060179195179018` im **C-XSC** System zur nächsten Rasterzahl $\widetilde{x}$, so ist $1 \oplus \widetilde{x}$ nicht die nach oben gerundete nächste Rasterzahl, sondern der exakte Wert $1 + \widetilde{x}$ wird, aus welchen Gründen auch immer, nach unten zur weiter entfernten nächsten Rasterzahl gerundet, so dass das Maschinenergebnis $1 \oplus \widetilde{x}$ nicht mehr maximal genau ist! Für die Fehlerabschätzung ergibt sich daraus die Forderung:

> Soll der exakte Funktionswert $f(\widetilde{x}) \in \mathbb{R}$ auf einem Rechner mit Hilfe einer Intervallfunktion in der Praxis garantiert eingeschlossen werden, so muss man bei der zur Implementierung dieser Intervallfunktion notwendigen Fehlerabschätzung voraussetzen, dass die Grundoperationen nur in hochgenauer Arithmetik ausgeführt werden. Alle nachfolgenden Beispiele zur Fehlerabschätzung beziehen sich daher in diesem Buch nur auf die hochgenaue Arithmetik!

2.5.2 Hochgenaue Arithmetik

Wir wählen wieder die bisherigen Bezeichnungen. A, B sind die Intervalloperanden mit den exakten Operanden $a \in A,\ b \in B$, wobei wir die reellen Intervalle A, B als Wertebereiche reeller Funktionen auffassen können. Wir nehmen jetzt an, dass diese Wertebereiche auf der Maschine fehlerhaft berechnet wurden. An Stelle von $a \in \mathbb{IR}$ wurde also der fehlerbehaftete Maschinenwert $\widetilde{a} \in S(2,53)$ bestimmt, und für alle $a \in A$ soll gelten: $|a - \widetilde{a}| \leq \Delta(a)$, d.h. die absolute Fehlerschranke $\Delta(a)$ sei bekannt. Falls $0 \notin A$ kann auch vorausgesetzt werden, dass die relative Fehlerschranke $\varepsilon(a)$ gegeben ist, d.h. dass für alle $a \in A$ gilt: $|a - \widetilde{a}| \leq \varepsilon(a) \cdot |a|$. Für die Maschinenzahlen $\widetilde{a}$ gilt: $\widetilde{a} \in \widetilde{A} := A + [-\Delta(a), +\Delta(a)]$. Ganz entsprechende Überlegungen gelten natürlich auch für $B, \widetilde{b}, b, \Delta(b)$ und $\varepsilon(b)$.

Für alle $a \in A$ und für alle $b \in B$ suchen wir jetzt absolute bzw. relative Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$, so dass für $\circ \in \{+, -, \cdot, /\}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ gilt:

$$|(a \circ b) - (\widetilde{a} \bullet \widetilde{b})| \leq \Delta(\circ) \qquad \text{bzw.} \qquad \left| \frac{(a \circ b) - (\widetilde{a} \bullet \widetilde{b})}{a \circ b} \right| \leq \varepsilon(\circ)$$

In den Abschnitten 2.3 und 2.4 wurden für die Fehlerschranken $\Delta(\circ)$ und $\varepsilon(\circ)$ die entsprechenden Formeln zusammengestellt, wobei zu unterscheiden war, ob die Operationsergebnisse $\widetilde{a} \bullet \widetilde{b}$ nur im Unterlaufbereich oder nur im normalisierten Bereich oder in beiden Bereichen liegen. Bei nur hochgenauer Arithmetik ist in den entsprechenden Formeln der auftretende Wert $\overline{\varepsilon}$ zu ersetzen durch $\overline{\varepsilon} = \varepsilon(h) = 2^{-52}$. Um beispielsweise nach (2.18) für $\Delta(\circ)$ wirklich eine Oberschranke zu erhalten, muss die Summe $\Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ)$ auf der Maschine mit Hilfe der **C-XSC** Funktion `addu` zur nächstgrößeren Maschinenzahl gerundet werden. Ganz entsprechend sind die in den Tabellen 2.1 und 2.2 angegebenen Ausdrücke für $\Delta_{dat}(\circ)$ durch gerichtetes Runden auszuwerten.

Um die Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$ auf der Maschine berechnen zu können, müssen die reellen Operandenintervalle $A, B \in I\,\mathbb{IR}$ durch Maschinenintervalle $\mathtt{A}, \mathtt{B} \in I\,R$ möglichst optimal eingeschlossen werden, d.h. es muss gelten: $A \subseteq \mathtt{A}$ und $B \subseteq \mathtt{B}$, und diese Maschinenintervalle sind dann Eingangsparameter der entsprechenden **C-XSC** Funktionen, mit denen die gesuchten Fehlerschranken $\Delta(\circ)$ bzw. $\varepsilon(\circ)$ zu berechnen sind. Ganz entsprechend muss man auch für die absoluten und relativen Fehlerschranken $\Delta(a), \Delta(b), \varepsilon(a), \varepsilon(b)$ möglichst kleine Rasterzahlen $\mathtt{dela}, \mathtt{delb}, \mathtt{epsa}, \mathtt{epsb} \in S(2,53)$ angeben, so dass gilt:

$$\Delta(a) \leq \mathtt{dela},\ \Delta(b) \leq \mathtt{delb},\ \varepsilon(a) \leq \mathtt{epsa},\ \varepsilon(b) \leq \mathtt{epsb}.$$

Jeweils eine dieser Rasterzahlen `dela,delb` und `epsa,epsb` ist dann wieder den entsprechenden **C-XSC** Funktionen als weitere Eingangsparameter zu übergeben.

Rückgabewerte der **C-XSC** Funktionen sind eine Oberschranke für $\Delta(\circ)$ bzw. für $\varepsilon(\circ)$ und ein Maschinenintervall $\mathtt{R} \in IR$, das eine Einschließung von $\mathtt{A} \circ \mathtt{B}$ liefert, d.h. $\mathtt{A} \circ \mathtt{B} \subseteq \mathtt{R} := \mathtt{A} \bullet \mathtt{B}; \quad \circ \in \{+, -, \cdot, /\}$ und $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$.

Anhand eines Beispiels soll jetzt gezeigt werden, wie man bei nur hochgenauer Arithmetik mit Hilfe der im Modul `abs_relh` implementierten **C-XSC** Funktionen eine Fehlerschranke des absoluten Fehlers berechnen kann, wenn der Term $1 + x$ auf der Maschine für alle reellen $x \in B = [1, 2.1]$ ausgewertet wird. Wir interpretieren B wieder als Wertebereich einer reellen Funktion und nehmen an, dass die Funktionswerte $x \in B$ hochgenau berechnet wurden, d.h. es soll für alle $x \in B$ gelten: $|x - \widetilde{x}| \leq \varepsilon(b) \cdot |x|$, mit $\varepsilon(b) = 2^{-52} = 2.22044604\ldots \cdot 10^{-16}$. Es wird also angenommen, dass auf der Maschine die Additionen $1 \oplus \widetilde{x}$ ausgeführt werden. Da der größte absolute Fehler gegeben ist durch $\Delta(x) := \varepsilon(b) \cdot 2.1$, liegen alle Maschinenzahlen $\widetilde{x}$ im Intervall $\widetilde{B} := B + [-\Delta(x), +\Delta(x)]$. Das erste Operandenintervall A ist jetzt ein Punktintervall, d.h. $A = \mathtt{A} = [1, 1] = \mathtt{interval(1)}$, mit $\Delta(a) = \varepsilon(a) = 0$.

```
#include "abs_relh.hpp" // Wegen abs_addh2(...)
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;
int main()
{
    interval A=interval(1), B,R;
    string("[1,2.1]") >> B; // B schließt [1,2.1] ein!
    real del, dela=0, epsb;
    cin >> RndUp; string("2.22044605E-16") >> epsb;
       // Die Funktion abs_addh2 aus dem Modul abs_relh
    abs_addh2(A,dela,B,epsb,R,del);
       // berechnet R und die absolute Fehlerschranke del.
    cout << "Absolute Fehlerschranke = " << del << endl;
    cout << "A+B enthalten in R = " << R << endl;
}
```

Das obige Programm `book_expl07.cpp` liefert die folgende Bildschirmausgabe:

```
Absolute Fehlerschranke = 1.154632E-015
A+B enthalten in R = [  2.000000,  3.100001]
```

Anmerkungen:

1. Es gelten wieder die ganz entsprechenden Anmerkungen 1,2,4 von Seite 32.

2. Der Vergleich mit dem absoluten Fehler `del` $= 5.773160 \cdot 10^{-16}$ von Seite 32 zeigt, dass der absolute Fehler `del` $= 1.154632 \cdot 10^{-15}$ bei jetzt nur hochgenauer Arithmetik etwa um den Faktor 2 größer ausfällt. Beachten Sie bitte, dass man die in diesem Beispiel angegebene relative Fehlerschranke $\varepsilon(b) = 2^{-52}$ auch ganz anders hätte wählen können. Sie muss jedenfalls nicht wie hier als $\varepsilon(b) = 2^{-52} = \varepsilon(h)$ festgelegt werden, nur weil die Arithmetik bzgl. der Grundoperationen als hochgenau vorausgesetzt wurde! Erinnern Sie sich bitte in diesem Zusammenhang, dass wir die reellen $x \in B$ auffassen wollten als die exakten Funktionswerte einer beliebigen Funktion und dass diese exakten Funktionswerte auf einem Rechner ausgewertet sein sollten und daher nur als fehlerbehaftete Maschinenzahlen $\widetilde{x}$ zur Verfügung stehen. Für alle $x \in B$ ist der relative Fehler $\varepsilon(b)$ definiert durch: $|x - \widetilde{x}| \leq \varepsilon(b) \cdot |x|$, und $\varepsilon(b)$ wurde in diesem Beispiel willkürlich gewählt als $\varepsilon(b) = 2^{-52}$.

3. Für `epsb` $= 0$ liefert das letzte Programmbeispiel die absolute Fehlerschranke `del` $\leq 6.883383 \cdot 10^{-16}$, d.h. für alle Maschinenzahlen $x = \widetilde{x} \in$ `B` gilt bei nur hochgenauer Arithmetik:

$$|(1 + \widetilde{x}) - (1 \oplus \widetilde{x})| \leq \Delta(+) \leq \texttt{del} \leq 6.883383 \cdot 10^{-16}$$

   Die Fehlerschranke `del` wurde dabei mit Hilfe der Funktion `abs_addh2` aus dem **C-XSC** Modul `abs_relh` berechnet.

Das **C-XSC** Modul `abs_relh` stellt $2 \cdot 4 \cdot 4 = 32$ Funktionen zur Verfügung, um bei einer nur hochgenauen Arithmetik für die vier Grundoperationen absolute oder relative Fehlerschranken zu berechnen. Der erste Faktor 2 berücksichtigt sowohl die absoluten als auch die relative Fehlerschranken. Der zweite Faktor 4 steht für die vier Grundoperationen, und der letzte Faktor 4 berücksichtigt die vier Zeilen in den Tabellen 2.1 und 2.2 auf den Seiten 24 und 27, wobei in jeder Zeile die Art der jeweiligen Fehlerschranken für die Intervalloperanden A, B festgelegt ist. In der jeweils dritten Zeile wird z.B. festgelegt, dass für den ersten Intervalloperanden A eine relative und für den zweiten Intervalloperanden B eine absolute Fehlerschranke vorgegeben ist.

Für die Namensgebung der 32 Funktionen aus dem Modul `abs_relh` zur Berechnung der absoluten oder relativen Fehlerschranken bei den Grundoperationen gelten fast die gleichen auf Seite 33 beschriebenen Vereinbarungen. Lediglich das vorletzte Namenszeichen muss ein `h` sein, womit die benutzte hochgenaue Arithmetik bei den Grundoperationen symbolisiert wird.

2.5.3 Zusammenstellung der Funktionen

Für eine maximal genaue bzw. hochgenaue Arithmetik stellen die Module `abs_relm` bzw. `abs_relh` jeweils 32 Funktionen zur Berechnung absoluter oder relativer Fehlerschranken bzgl. der vier Maschinenoperationen $\{\oplus, \ominus, \odot, \oslash\}$ zur Verfügung. In den vorhergehenden Unterabschnitten wurde die Anwendung einiger Funktionen in mehreren Beispielen ausführlich beschrieben. Zur besseren Übersicht sind alle diese Funktionen in den nachfolgenden Tabellen zusammengestellt:

Abs. Schranken bei maximal genauer Arithmetik			
$\oplus$	$\ominus$	$\odot$	$\oslash$
`abs_addm1()`	`abs_subm1()`	`abs_mulm1()`	`abs_divm1()`
`abs_addm2()`	`abs_subm2()`	`abs_mulm2()`	`abs_divm2()`
`abs_addm3()`	`abs_subm3()`	`abs_mulm3()`	`abs_divm3()`
`abs_addm4()`	`abs_subm4()`	`abs_mulm4()`	`abs_divm4()`
Rel. Schranken bei maximal genauer Arithmetik			
$\oplus$	$\ominus$	$\odot$	$\oslash$
`rel_addm1()`	`rel_subm1()`	`rel_mulm1()`	`rel_divm1()`
`rel_addm2()`	`rel_subm2()`	`rel_mulm2()`	`rel_divm2()`
`rel_addm3()`	`rel_subm3()`	`rel_mulm3()`	`rel_divm3()`
`rel_addm4()`	`rel_subm4()`	`rel_mulm4()`	`rel_divm4()`

Tabelle 2.3: Berechnung absoluter u. relativer Fehlerschranken

Eine ausführliche Beschreibung zum Aufruf der Funktion `abs_addm2()` findet man im Programm auf Seite 31. Die jeweils letzte Ziffer der obigen Funktionsnamen legt die Bedeutung der jeweils zweiten und vierten Funktionsparameter fest. Die Ziffer 2 im Namen der Funktion `abs_addm2()` besagt nach Zeile 2 in Tabelle 2.1 auf Seite 24, dass die Fehlerschranke der gestörten Werte des ersten Operanden durch die absolute Fehlerschranke $\Delta(a)$ und dass die Fehlerschranke der gestörten Werte des zweiten Operanden durch die relative Fehlerschranke $\varepsilon(b)$ gegeben ist.

Der jeweils vorletzte Parameter aller Funktionen liefert als Maschinenintervall eine optimale Einschließung aller exakten Ergebnisse $a \circ b$, mit $a \in \mathtt{A}$, $b \in \mathtt{B}$ und $\circ \in \{+, -, \cdot, /\}$. Das Maschinenintervall `A` ist dabei als erster Funktionsparameter eine Einschließung aller exakten Werte a des ersten Operanden, und das Maschinen-

intervall `B` ist als dritter Funktionsparameter eine Einschließung aller exakten Werte b des zweite Operanden bei der jeweils betrachteten Grundoperation.

Der jeweils letzte Funktionsparameter liefert dann eine absolute oder relative Gesamtfehlerschranke $\Delta(\circ)$ oder $\varepsilon(\circ)$, vgl. dazu die entsprechenden Definitionen auf Seite 30.

Werden die Maschinenoperationen $\{\oplus, \ominus, \odot, \oslash\}$ in nur hochgenauer Arithmetik ausgeführt, so können die auf Seite 36 definierten absoluten oder relativen Gesamtfehlerschranken $\Delta(\circ)$ oder $\varepsilon(\circ)$ mit Hilfe der in folgender Tabelle zusammengestellten Funktionen aus dem Modul `abs_relh` berechnet werden:

Abs. Schranken bei nur hochgenauer Arithmetik			
$\oplus$	$\ominus$	$\odot$	$\oslash$
`abs_addh1()`	`abs_subh1()`	`abs_mulh1()`	`abs_divh1()`
`abs_addh2()`	`abs_subh2()`	`abs_mulh2()`	`abs_divh2()`
`abs_addh3()`	`abs_subh3()`	`abs_mulh3()`	`abs_divh3()`
`abs_addh4()`	`abs_subh4()`	`abs_mulh4()`	`abs_divh4()`
Rel. Schranken bei nur hochgenauer Arithmetik			
$\oplus$	$\ominus$	$\odot$	$\oslash$
`rel_addh1()`	`rel_subh1()`	`rel_mulh1()`	`rel_divh1()`
`rel_addh2()`	`rel_subh2()`	`rel_mulh2()`	`rel_divh2()`
`rel_addh3()`	`rel_subh3()`	`rel_mulh3()`	`rel_divh3()`
`rel_addh4()`	`rel_subh4()`	`rel_mulh4()`	`rel_divh4()`

Tabelle 2.4: Berechnung absoluter u. relativer Fehlerschranken

Eine Beschreibung der Funktion `abs_addh2()` findet man auf Seite 37.

Ist bei einer Multiplikation der erste Operand eingeschlossen durch `A` $= [2^k, 2^k]$, mit $k \in \mathbb{Z}$, und liegt kein Maschinenprodukt im Unterlaufbereich, so wird der absolute oder relative Rundungsfehler auf Null gesetzt. Ist bei der Division der zweite Operand `B` $= [2^k, 2^k]$, mit $k \in \mathbb{Z}$, und liegt kein Quotient im Unterlaufbereich U, so wird der Rundungsfehler ebenfalls auf Null gesetzt. In den Funktionen aus der Tabelle 2.3 sind damit bei Multiplikation und Division die in Abschnitt 3.1 für den normalisierten Bereich angegebenen Sätze eingearbeitet. Bei den Funktionen der Tabelle 2.4 sind jedoch alle vier Sätze des Abschnitts 3.1 bereits vollständig berück-

sichtigt!

2.5.4 Beispiel

In den bisherigen Beispielen der Abschnitte 2.5.1 und 2.5.2 haben wir jeweils Terme $T(x)$ mit nur einer Grundoperation betrachtet. Wir wollen jetzt für den Term

$$T(x) := \frac{1+x}{1-x}$$

zunächst den absoluten Fehler unter folgenden Voraussetzungen berechnen:

1. Die in $T(x)$ auftretenden Grundoperationen $+, -, /$ sollen auf der Maschine in nur hochgenauer Arithmetik ausgeführt werden.

2. Für die exakten Werte $x \in I\!R$ soll gelten: $x \in \mathtt{X} := [-10, +0.5]$.

3. Da nicht alle $x \in \mathtt{X}$ als Maschinenzahlen $\widetilde{x} \in S(2, 53)$ darstellbar sind, wird vorausgesetzt, dass ein solches $x \in \mathtt{X}$ stets zur nächsten Rasterzahl $\widetilde{x}$ gerundet wird. Der durch diese Rundung bedingte absolute Fehler $|x - \widetilde{x}|$ lässt sich dann abschätzen durch $|x - \widetilde{x}| \leq |\mathtt{X}| \cdot \varepsilon(m) = 10 \cdot \varepsilon(m) = 10 \cdot 2^{-53} = \Delta(x)$. Für alle $x \in \mathtt{X}$ soll damit gelten:

$$|x - \widetilde{x}| \leq \Delta(x) = 10 \cdot 2^{-53} = 1.11022302\ldots \cdot 10^{-15}, \quad x \in \mathtt{X} = [-10, +0.5]$$

Bezeichnen wir mit $\widetilde{T}(\widetilde{x}) := (\widetilde{x} \ominus 1) \oslash (\widetilde{x} \oplus 1)$ den auf der Maschine ausgewerteten Term $T(x)$, so berechnet das nachfolgende Programm eine Oberschranke `del` des dabei aufgetretenen absoluten Fehlers $|T(x) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del}$.

```
#include "abs_relh.hpp" // Wegen abs_addh1, ...
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
int main()
{
    interval One=interval(1), X=interval(-10,0.5),A,B,R;
    real delx,dela,delb,del;
    cin >> RndUp; string("1.11022303E-15") >> delx;
    abs_addh1(One,0,X,delx,A,dela); // dela: abs. Fehler für 1+x
    abs_subh1(One,0,X,delx,B,delb); // delb: abs. Fehler für 1-x
    abs_divh1(A,dela,B,delb,R,del);
                    // del : absoluter Fehler bzgl. (1+x)/(1-x)
    cout << "Absolute Fehlerschranke = " << del << endl;
    cout << "(1+x)/(1-x) enthalten in:  " << R << endl;
}
```

Das umseitige Programm `book_expl08.cpp` liefert die folgende Bildschirmausgabe:

```
Absolute Fehlerschranke = 1.381118E-013
(1+x)/(1-x) enthalten in:  [-18.000000,  3.000000]
```

Anmerkungen:

1. Für alle $x \in \mathtt{X} = [-10, +0.5]$ gilt damit für die Abschätzung des absoluten Fehlers $|T(x) - \widetilde{T}(\widetilde{x})|$:

$$|T(x) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del} \leq 1.381118 \cdot 10^{-13}$$

2. Zur Abschätzung des absoluten Fehlers müssen im umseitigen Programm die entsprechenden Funktionen `abs_addh1`, `abs_subh1` und `abs_divh1` lediglich in der Reihenfolge der im Term durchzuführenden Grundoperationen angegeben werden.

3. Erst beim jetzigen Term $T(x)$, in dem drei Grundoperationen nacheinander auszuführen sind, wird deutlich, wie nützlich es ist, dass jede einzelne Funktion `abs_addh1`, `abs_subh1` und `abs_divh1` als vorletzten Parameter jeweils eine Einschließung der exakten Verknüpfungsergebnisse liefert, auf die bei der Abschätzung des jeweils nächsten Fehlers zurückgegriffen werden muss.

4. Wählt man $\mathtt{delx} = 0$, so liefert das Programm die natürlich kleinere absolute Fehlerschranke

$$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del} \leq 9.592327 \cdot 10^{-14} \quad \text{für alle } \widetilde{x} \in \mathtt{X} = [-10, +0.5]$$

   Beachten Sie bitte, dass jetzt das Intervall $\mathtt{X} = [-10, +0.5]$ wegen $\mathtt{delx} = 0$ nur die exakten Maschinenzahlen $\widetilde{x} \in \mathtt{X}$ enthält und dass sich obige Fehlerschranke nur auf diese Maschinenzahlen bezieht!

5. Wenn man jetzt unter der gleichen Voraussetzung $\mathtt{delx} = 0$ das Intervall $\mathtt{X}$ in die beiden Teilintervalle $\mathtt{X_1} = [-10, -5]$ und $\mathtt{X_2} = [-5, +0.5]$ unterteilt und für beide Teilintervalle den jeweiligen absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ extra berechnet, so erhält man die Ergebnisse:

$$\begin{aligned} |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| &\leq \mathtt{del1} \leq 1.276756 \cdot 10^{-15} \quad \text{für alle } \widetilde{x} \in \mathtt{X_1} = [-10, -5] \\ |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| &\leq \mathtt{del2} \leq 2.486900 \cdot 10^{-14} \quad \text{für alle } \widetilde{x} \in \mathtt{X_2} = [-5, +0.5] \end{aligned}$$

Nach Punkt 5 der letzten Anmerkungen gilt: $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq 2.486900 \cdot 10^{-14}$ für alle $\widetilde{x} \in \mathtt{X} = [-10, +0.5]$. Vergleicht man mit der absoluten Fehlerschranke aus Punkt

4, so erhält man eine etwa um den Faktor 4 kleinere Fehlerschranke, wenn man das Intervall `X` $= [-10, +0.5]$ in nur zwei Teilintervalle $\texttt{X}_1$ und $\texttt{X}_2$ unterteilt.

Im nächsten Programm wollen wir für den gleichen Term $T(x) = (1+x)/(1-x)$ über dem gleichen Intervall `X` $= [-10, +0.5]$ unter der Voraussetzung $\Delta(x) = 0$ den absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ möglichst optimal abschätzen, indem wir das Intervall `X` in hinreichend viele Teilintervalle `Xi` zerlegen und für jedes Teilintervall `Xi` den absoluten Fehler einzeln abschätzen. Der absolute Fehler über dem Gesamtintervall `X` ist dann das Maximum der absoluten Fehler über allen Teilintervallen `Xi`.

```
#include "abs_relh.hpp" // Wegen abs_addh1, abs_subh1, ...
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Xi, const real& delx)
{
    interval One=interval(1),A,B,R;
    real dela,delb,del;
    abs_addh1(One,0,Xi,delx,A,dela); // absl. Fehler bzgl. 1+x
    abs_subh1(One,0,Xi,delx,B,delb); // absl. Fehler bzgl. 1-x
    abs_divh1(A,dela,B,delb,R,del);  // absl. Fehler bzgl. (1+x)/(1-x)
    return del;  // Rückgabe einer Schranke del des absoluten Fehlers.
}

int main()
{
    interval X;
    real bnd, diam = 1e-5, delx=0;
    cout << "Intervall X = ? "; cin >> X;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Absolute Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl09.cpp` liefert bei Eingabe von `X` $= [-10, 0.5]$ den Bildschirmausdruck:

```
Absolute Fehlerschranke = 1.998409E-015
```

Anmerkungen:

1. Wegen $\Delta(x) =$ `delx` $= 0$ bezieht sich die letzte Fehlerschranke nur auf die Maschinenzahlen $\widetilde{x} \in$ `X`. Für den absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ gilt daher bei nur hochgenauer Arithmetik die Abschätzung:

   $$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq 1.998409 \cdot 10^{-15} \quad \text{für alle } \widetilde{x} \in \texttt{X} = [-10, +0.5]$$

   Vergleicht man mit den Fehlerschranken aus Punkt 5 von Seite 43, so erkennt man, das der absolute Fehler durch die sehr feine Intervallunterteilung etwa um den Faktor 10 verkleinert werden konnte.

2. Beachten Sie bitte, dass in der Funktion `main()` der Manipulator `RndUp` in der letzten Zeile nicht vergessen werden darf, da durch ihn die binäre Fehlerschranke `bnd` zur nächstgrößeren Dezimalzahl des gewählten Ausgabeformats aufgerundet wird. Bei fehlendem Manipulator `RndUp` erfolgt die Rundung zur nächsten Dezimalzahl, was eventuell mit einer Abrundung verbunden ist. In einem solchen Fall kann der ausgegebene Dezimalwert dann nicht mehr als garantierte Oberschranke des absoluten Fehlers angesehen werden!

3. Die Funktion `T_x()` liefert für $T(x) := (1 + x)/(1 - x)$ eine Oberschranke des absoluten Fehlers $|T(x) - \widetilde{T}(\widetilde{x})|$ für alle $x \in$ `X`, wenn für die gestörten Maschinenzahlen $\widetilde{x}$ gilt: $|x - \widetilde{x}| \leq \Delta(x) =$ `delx`.

4. Die Funktion `Max_bnd_Xi()` aus dem **C-XSC** Modul `bnd_util` zerlegt das Intervall `X` in Teilintervalle `Xi`, deren Durchmesser durch den übergebenen Wert `diam` bestimmt ist, wobei `diam` $= 10^{-5}$ meist eine gute Wahl ist. Mit Hilfe von `T_x()` wird für jedes Teilintervall `Xi` die durch `T_x()` bestimmte Fehlerschranke abgeschätzt und das Maximum all dieser Fehlerschranken in der `real`-Variablen `bnd` gespeichert. Der hier an `Max_bnd_Xi()` übergebene Parameter `delx` wird intern weiter an `T_x()` übergeben und bestimmt die absolute Fehlerschranke der gestörten Werte $\widetilde{x}$, d.h. es gilt: $|x - \widetilde{x}| \leq \Delta(x) =$ `delx` für alle $x \in$ `X`.

Wir wollen jetzt noch für den gleichen Term $T(x) := (1+x)/(1-x)$ eine Oberschranke des relativen Fehlers $|(T(x)-\widetilde{T}(\widetilde{x}))/T(x)|$ für alle $x \in \texttt{X} = [-10, +0.5]$ berechnen. Vorausgesetzt wird wieder $\Delta(x) = 0$, so dass der relative Fehler $(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})$ jetzt also nur noch für alle Maschinenzahlen $\widetilde{x} \in \texttt{X} = [-10, +0.5]$ abzuschätzen ist. Wegen $T(-1) = \widetilde{T}(-1) = 0$ ist der relative Fehler für $\widetilde{x} = -1$ unbestimmt und kann deshalb auf Null gesetzt werden. Die Fehlerabschätzung muss daher nur noch in den beiden Intervallen `X1` $= [-10, \texttt{pred}(-1)]$ und `X2` $= [\texttt{succ}(-1), +0.5]$ vorgenommen werden.

Für das Intervall `X1` muss das auf Seite 44 angegebene Programm im Wesentlichen nur in der vorletzten Zeile der Funktion `T_x()` zur Abschätzung des relativen Fehlers bei der Division abgeändert werden:

```
#include "abs_relh.hpp" // Wegen abs_addh1, abs_subh1, ...
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Xi, const real& delx)
{
    interval One=interval(1),A,B,R;
    real dela,delb,eps;
    abs_addh1(One,0,Xi,delx,A,dela); // abs. Fehler bzgl. 1+x
    abs_subh1(One,0,Xi,delx,B,delb); // abs. Fehler bzgl. 1-x
    rel_divh1(A,dela,B,delb,R,eps);  // rel. Fehler bzgl. (1+x)/(1-x)
    return eps;
}

int main()
{
    interval X1 = interval(-10,pred(-1));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X1,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl10.cpp` liefert die folgende Bildschirmausgabe:

```
Relative Fehlerschranke = 1.999995E-005
```

Anmerkungen:

1. Wegen $\Delta(x) = \texttt{delx} = 0$ bezieht sich die obige Fehlerschranke nur auf die Maschinenzahlen $\widetilde{x} \in \texttt{X1} = [-10, \texttt{pred}(-1)]$. Für den gesuchten relativen Fehler $|(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})|$ gilt daher bei vorausgesetzter hochgenauer Arithmetik die Abschätzung:

$$\left| \frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})} \right| \leq 1.999995 \cdot 10^{-5} \quad \text{für alle } \widetilde{x} \in \texttt{X1} = [-10, \texttt{pred}(-1)]$$

2. Ändert man im obigen Programm in der Funktion `main()` die drei ersten Zeilen wie folgt

```
interval X2 = interval(succ(-1),+0.5);
real bnd, diam = 1e-5, delx=0;
Max_bnd_Xi(T_x,X2,delx,diam,bnd);
```

so erhält man für den relativen Fehler $|(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})|$ die Abschätzung:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq 2.000001 \cdot 10^{-5} \quad \text{für alle } \widetilde{x} \in \texttt{X2} = [\texttt{succ}(-1), +0.5]$$

3. Im gesamten Intervall $\texttt{X} = [-10, +0.5]$ gilt dann für den relativen Fehler die Abschätzung:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq 2.000001 \cdot 10^{-5} \quad \text{für alle } \widetilde{x} \in \texttt{X} = [-10, +0.5]$$

Wir hatten dabei $\Delta(x) = \texttt{delx} = 0$ vorausgesetzt, so dass für alle $x \in \texttt{X}$ gilt: $x = \widetilde{x}$. Die obige Fehlerabschätzung bezieht sich daher nur auf die Rasterzahlen $\widetilde{x} \in \texttt{X} = [-10, +0.5]$, d.h. $\widetilde{x} \in \texttt{X} \cap S(2, 53)$.

4. Die oben berechnete relative Fehlerschranke $2.000001 \cdot 10^{-5}$ kann noch um viele Größenordnungen verbessert werden, wenn man die Abschätzung der absoluten Fehler für die Terme $1 + x$ und $1 - x$ nach Abschnitt 3.2 weiter verbessert. Die verhältnismäßig große relative Fehlerschranke wird dadurch begründet, dass $T(\widetilde{x})$ in der Nähe seiner Nullstelle $\widetilde{x} = -1$ auszuwerten ist und dass dabei Quotienten mit fehlerbehafteten Operanden zu bilden sind. In Abschnitt 3.2 wird gezeigt, dass diese Operanden in der Umgebung der Nullstelle teilweise rundungsfehlerfrei dargestellt werden können, womit dann der relative Fehler deutlich verbessert werden kann.

5. Startet man das obige Beispielprogramm `book_expl10.cpp` mit Hilfe der nach Abschnitt 3.1 jetzt schon optimierten Funktionen `abs_addh1()`, `abs_subh1()` und `rel_divh1()` aus dem Modul `abs_relh.hpp`, so erhält man die um viele Größenordnungen verbesserte relative Fehlerschranke: $4.440932 \cdot 10^{-16}$.

Wir haben bis jetzt schon erkannt, dass eine optimale Fehlerabschätzung mit großer Sorgfalt durchzuführen ist. Dabei muss man gegebenenfalls zu breite Argumentintervalle in möglichst schmale Teilintervalle zerlegen oder für spezielle Terme nach Abschnitt 3.2 gesonderte Abschätzungen vornehmen.

Kapitel 3

Verbesserte Fehlerschranken

Nehmen wir an, dass die vier Grundoperationen $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$ auf der Maschine in maximal genauer Arithmetik durchgeführt werden, dann gilt nach (1.14) im normalisierten Bereich, d.h. $\widetilde{a} \bullet \widetilde{b} \notin U = (-\texttt{MinReal}, +\texttt{MinReal})$ und damit auch $\widetilde{a} \circ \widetilde{b} \notin U$, für die Fehlerschranke des absoluten Fehlers:

$$|(\widetilde{a} \circ \widetilde{b}) - (\widetilde{a} \bullet \widetilde{b})| \leq \varepsilon(m) \cdot |\widetilde{a} \circ \widetilde{b}|, \qquad \text{mit: } \circ \in \{+, -, \cdot, /\} \tag{3.1}$$

Wir betrachten dazu die folgenden Beispiele:

1. $\widetilde{a} \odot 2^n,\ n \in I\!N_0 \quad \longrightarrow \quad |\widetilde{a} \odot 2^n - \widetilde{a} \cdot 2^n| = 0.$
2. $2 \oplus 0.75 \quad \longrightarrow \quad |2 \oplus 0.75 - (2 + 0.75)| = 0.$
3. $1 \oplus 2^{-1022} \quad \longrightarrow \quad |1 \oplus 2^{-1022} - (1 + 2^{-1022})| = 2^{-1022} = \texttt{MinReal}.$

Die beiden ersten Beispiele liefern den absoluten Fehler 0, und das dritte Beispiel ergibt bei maximal genauer Arithmetik wegen $1 \oplus 2^{-1022} = 1$ den absoluten Fehler $|1 \oplus 2^{-1022} - (1 + 2^{-1022})| = \texttt{MinReal}$. Bei allen drei Beispielen ist die absolute Fehlerschranke damit kleiner als die in (3.1) angegebene Oberschranke $\varepsilon(m) \cdot |\widetilde{a} \circ \widetilde{b}|$. Es wäre also zur Berechnung möglichst kleiner Fehlerschranken wünschenswert, Sätze zu formulieren, die in Spezialfällen, d.h. bei speziellen Operanden, rundungsfehlerfreie Ergebnisse liefern. Diese Sätze werden im folgenden Abschnitt zusammengestellt und sind bei der automatischen Fehlerabschätzung entsprechend zu berücksichtigen. Im folgenden Abschnitt findet man zu speziellen Operanden absolute Fehlerschranken, die jedoch kleiner ausfallen als die rechte Seite in (3.1).

3.1 Rundungsfehlerfreie Operationen

Für die automatische Fehlerabschätzung werden Voraussetzungen für die einschließenden Operandenintervalle $\widetilde{A}$ und $\widetilde{B}$ benötigt, die garantieren, dass sämtliche Operationen $\widetilde{a} \circ \widetilde{b}$, mit $\widetilde{a} \in \widetilde{A} \cap S(2,53)$ und $\widetilde{b} \in \widetilde{B} \cap S(2,53)$ auf der Maschine rundungsfehlerfrei durchgeführt werden können, d.h.

$$\widetilde{a} \circ \widetilde{b} = \widetilde{a} \bullet \widetilde{b} \quad \forall\, \widetilde{a} \in \widetilde{A} \cap S(2,53) \text{ und } \forall\, \widetilde{b} \in \widetilde{B} \cap S(2,53)$$

mit $\circ \in \{+,-,\cdot,/\}$ und $\bullet \in \{\oplus,\ominus,\odot,\oslash\}$. Solche Voraussetzungen werden in den folgenden Sätzen angegeben. Mit $IS := \{[\underline{a},\overline{a}] \mid \underline{a},\overline{a} \in S(2,53),\ \underline{a} \le \overline{a}\}$ bezeichnen wir dabei die Menge der Maschinenintervalle. Die Funktionen $z_{lead}(x)$ und $z_{trail}(x)$ liefern für eine Maschinenzahl $x = (-1)^{s(x)} \cdot m(x) \cdot 2^{e(x)} \in S(2,53)$ die Anzahl der führenden bzw. abschließenden Nullen in der Mantisse $m(x)$. Für normalisierte Zahlen $x \in S(2,53)$ gilt nach (1.4) $z_{lead}(x) = 0$, da die führende Mantissenziffer stets 1 ist, und nach (1.5) gilt für alle denormalisierte Zahlen $z_{lead}(x) \ge 1$, da die führende Mantissenziffer stets Null ist. Die kleinste positive denormalisierte Zahl `minreal` ist nach Seite 5 gegeben durch $0.00\ldots001 \cdot 2^{-1022}$, so dass gilt: $z_{lead}(\texttt{minreal}) = 52$. Nach Seite 6 besitzt die normalisierte Zahl -44 die Mantisse $m(-44) = 1.01100\ldots00$ mit $53-4 = 49$ abschließenden Mantissenziffern Null, d.h. $z_{trail}(-44) = 49$. Für die Zahl 0 gilt: $z_{lead}(0) = 0$ und $z_{trail}(0) = 53$.

In der Darstellung $x = (-1)^{s(x)} \cdot m(x) \cdot 2^{e(x)} \in S(2,53)$ ist $e(x)$ der ganzzahlige Exponent zur Basis 2. Beachten Sie, dass nach Seite 6 die Werte `ex = expo(x)` und $e(x)$ verschieden sind und für normalisierte Zahlen die Beziehung `ex` $= e(x)+1$ gilt. Im denormalisierten Bereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$ gilt: $e(x) = -1022$.

In den folgenden Sätzen wird außerdem die Funktion $e_{IS} : IS \mapsto \mathbb{N}$ benutzt, die mit einem Maschinenintervall $X \in IS$ definiert ist durch:

$$e_{IS}(X) = \begin{cases} e(\langle X \rangle), & \text{falls } X \text{ echtes Maschinen-Intervall} \\ e(x) + z_{trail}(x), & \text{falls } X = [x,x] \text{ Maschinen-Punktintervall} \end{cases} \tag{3.2}$$

$\langle X \rangle$ bedeutet dabei das Minimum aller Betragselemente von $X \in IX$, d.h. $\langle X \rangle$ und x sind beides Maschinenzahlen.

Satz 3.1.1 (Addition) Ist eine der Bedingungen

1. $\mathrm{Inf}(\widetilde{A}/\widetilde{B}) \ge -2$ und $\mathrm{Sup}(\widetilde{A}/\widetilde{B}) \le -\frac{1}{2}$
2. $e(|\widetilde{A} + \widetilde{B}|) \le \min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\}$
3. $\widetilde{A} = [0,0]$ oder $\widetilde{B} = [0,0]$

erfüllt, dann gilt: $\widetilde{a} + \widetilde{b} = \widetilde{a} \oplus \widetilde{b}\ \ \forall\, \widetilde{a} \in \widetilde{A} \cap S(2,53)$ und $\forall\, \widetilde{b} \in \widetilde{B} \cap S(2,53)$.

Satz 3.1.2 (Subtraktion) Ist eine der Bedingungen

1. $\mathrm{Inf}(\widetilde{A}/\widetilde{B}) \geq \frac{1}{2}$ und $\mathrm{Sup}(\widetilde{A}/\widetilde{B}) \leq 2$
2. $e(|\widetilde{A} - \widetilde{B}|) \leq \min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\}$
3. $\widetilde{A} = [0,0]$ oder $\widetilde{B} = [0,0]$

erfüllt, dann gilt: $\widetilde{a} - \widetilde{b} = \widetilde{a} \ominus \widetilde{b} \;\; \forall \widetilde{a} \in \widetilde{A} \cap S(2,53)$ und $\forall \widetilde{b} \in \widetilde{B} \cap S(2,53)$.

Satz 3.1.3 (Multiplikation) Ist eine der Bedingungen

1. $\widetilde{A} = [\widetilde{a}, \widetilde{a}],\ \widetilde{a} = 2^k,\ k \in \mathbb{Z}$;
   falls $(\widetilde{A} \cdot \widetilde{B}) \cap U \neq \emptyset$, muss zusätzlich $k \geq -1022 - e_{IS}(\widetilde{B})$ sein;
2. $\widetilde{B} = [\widetilde{b}, \widetilde{b}],\ \widetilde{b} = 2^k,\ k \in \mathbb{Z}$;
   falls $(\widetilde{A} \cdot \widetilde{B}) \cap U \neq \emptyset$, muss zusätzlich $k \geq -1022 - e_{IS}(\widetilde{A})$ sein;
3. $\widetilde{A} = [\widetilde{a}, \widetilde{a}],\ \widetilde{B} = [\widetilde{b}, \widetilde{b}],\ z_{lead}(\widetilde{a}) + z_{trail}(\widetilde{a}) + z_{lead}(\widetilde{b}) + z_{trail}(\widetilde{b}) \geq 53$

erfüllt, dann gilt: $\widetilde{a} \cdot \widetilde{b} = \widetilde{a} \odot \widetilde{b} \;\; \forall \widetilde{a} \in \widetilde{A} \cap S(2,53)$ und $\forall \widetilde{b} \in \widetilde{B} \cap S(2,53)$.

Satz 3.1.4 (Division) Ist die Bedingung

$\widetilde{B} = [\widetilde{b}, \widetilde{b}],\ \widetilde{b} = 2^k,\ k \in \mathbb{Z}$ erfüllt und ist im Falle
$(\widetilde{A}/2^k) \cap U \neq \emptyset$ zusätzlich $k \leq 1022 + e_{IS}(\widetilde{A})$ erfüllt,

so gilt: $\widetilde{a}/\widetilde{b} = \widetilde{a} \oslash \widetilde{b} \;\; \forall \widetilde{a} \in \widetilde{A} \cap S(2,53)$.

Die Beweise dieser vier Sätze findet man wieder in [3]. Um bzgl. der Addition in Satz 3.1.1 die zweite Bedingung $e(|\widetilde{A} + \widetilde{B}|) \leq \min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\}$ auf dem Rechner auszuwerten, muss u.a. $\widetilde{A} + \widetilde{B}$ berechnet werden. Bei Anwendung der in **C-XSC** implementierten Intervallarithmetik erhält man die Einschließung $\widetilde{A} + \widetilde{B} \subseteq \widetilde{A} \oplus \widetilde{B}$ und daher $e(|\widetilde{A} + \widetilde{B}|) \leq e(|\widetilde{A} \oplus \widetilde{B}|)$, so dass in Satz 3.1.1 die zweite Bedingung auch wie folgt formuliert werden kann:

2a. $e(|\widetilde{A} \oplus \widetilde{B}|) \leq \min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\}$

Mit der Bedingung 2a. ist also auch die Bedingung 2. erfüllt, und damit kann die Maschinenaddition rundungsfehlerfrei erfolgen. Ganz ähnliche Überlegungen zeigen, dass bei der Maschinenauswertung in den Bedingungen der anderen drei Sätze die dort auftretenden exakten Grundoperationen durch entsprechende Intervalloperationen ersetzt werden können.

3.1.1 Beispiele

Beispiel 1 (Division) In diesem Beispiel soll ein punktförmiger Operand $\widetilde{A}$ aus dem normalisierten Bereich mit einer Mantisse, die keine abschließende Null besitzt, so durch $\widetilde{B} = 2^k,\ k \in I\!N$ dividiert werden, dass der Quotient $\widetilde{A}/2^k$ in den Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$ fällt. Da jedoch in U die Mantissenbreite der denormalisierten Zahlen kleiner ist als 53, kann $\widetilde{A}/2^k$ in U nicht mehr exakt dargestellt werden, und es gilt mit $\widetilde{A} = [\mathtt{a}, \mathtt{a}],\ \mathtt{a} \in S(2, 53)$:

$$\mathtt{a}/2^k \neq \mathtt{a} \oslash 2^k,$$

so dass die Division auf der Maschine nicht rundungsfehlerfrei erfolgen kann. Das Beispiel soll zeigen, dass die Voraussetzung des Satzes 3.1.4 in diesem Fall nicht erfüllt ist! Wir wählen a als Vorgänger der normalisierten Zahl 2^{-1021}, was in einem **C-XSC** Programm durch folgende Anweisungen realisiert werden kann:

```
real y = comp(0.5,-1020);   // y = 1.000...000*2^(-1021)
real a = pred(y);           // a = 1.111...111*2^(-1022)
```

Es gilt also $z_{trail}(\mathtt{a}) = 0,\ e(\mathtt{a}) = -1022$ und nach (3.2) von Seite 50 folgt:

$$e_{IS}(\widetilde{A}) = e(\mathtt{a}) + z_{trail}(\mathtt{a}) = -1022 + 0 = -1022$$

Wählt man jetzt z.B. $k = 20$, so lautet damit die Voraussetzung in Satz 3.1.4:

$$k = 20 \leq 1022 + (-1022) = 0,$$

wonach die rundungsfehlerfreie Division nach Satz 3.1.4 nicht garantiert werden kann und auch nicht möglich ist, wovon wir uns ja schon oben überzeugt hatten.

In den folgenden Beispielen zur Division betrachten wir $\widetilde{A}$ erst als echtes Intervall und danach als Punktintervall mit jeweils gleicher Untergrenze $4.296875 \cdot 10^{-2}$ und zeigen dabei, dass in beiden Fällen die jeweiligen Bedingungen für eine rundungsfehlerfrei Division durchaus verschieden ausfallen.

Beispiel 2 (Division) Wir wählen $\widetilde{A} = [4.296875 \cdot 10^{-2}, 10]$ und $\widetilde{B} = 2^k,\ k \in I\!N$, und fragen nach der Bedingung, die nach Satz 3.1.4 erfüllt sein muss, damit für alle $\widetilde{a} \in \widetilde{A} \cap S(2, 53)$ die Division $\widetilde{a} \oslash 2^k$ rundungsfehlerfrei durchgeführt werden kann. Zunächst gilt nach Seite 6: $4.296875 \cdot 10^{-2} = 1.375 \cdot 2^{-5} = 1.01100\ldots00_{bin} \cdot 2^{-5}$, und (3.2) liefert $e_{IS}(\widetilde{A}) = e(\langle\widetilde{A}\rangle) = e(4.296875 \cdot 10^{-2}) = e(1.01100\ldots00_{bin} \cdot 2^{-5}) = -5$. Die Bedingung für eine rundungsfehlerfreie Division lautet dann nach Satz 3.1.4

$$k \leq 1022 + e_{IS}(\widetilde{A}) = 1022 + (-5) = 1017 \tag{3.3}$$

Wegen $e_{IS}(\widetilde{A}) = e(\langle\widetilde{A}\rangle) = -5$ verhindert die obige Bedingung, dass die Quotienten $\widetilde{a}/2^k$ in den Unterlaufbereich fallen, womit dann die Divisionsergebnisse natürlich rundungsfehlerfrei bleiben!

Beispiel 3 (Division) Mit $\mathtt{a} = 4.296875 \cdot 10^{-2} \in S(2,53)$ wählen wir jetzt das Punktintervall $\widetilde{A} = [\mathtt{a},\mathtt{a}]$ und $\widetilde{B} = [2^k, 2^k],\ k \in I\!N$, und fragen nach der Bedingung, die nach Satz 3.1.4 erfüllt sein muss, damit die Division $\mathtt{a} \oslash 2^k$ rundungsfehlerfrei durchgeführt werden kann. Es gilt $4.296875 \cdot 10^{-2} = 1.01100\ldots00_{bin} \cdot 2^{-5}$, und (3.2) liefert: $e_{IS}(\widetilde{A}) = e(\mathtt{a}) + z_{trail}(\mathtt{a}) = -5 + 49 = 44$. Die Bedingung für eine rundungsfehlerfreie Division lautet dann nach Satz 3.1.4

$$k \leq 1022 + e_{IS}(\widetilde{A}) = 1022 + 44 = 1066 \tag{3.4}$$

Da $2^k \in S(2,53)$ erfüllt sein muss, gilt $k \leq +1023$, so dass die Bedingung (3.4) keine wirkliche Einschränkung bedeutet und damit die Division $\mathtt{a} \oslash 2^k$ für alle möglichen $k \leq +1023$ rundungsfehlerfrei durchgeführt werden kann. Beachten Sie bitte, dass die rundungsfehlerfreien Quotienten $\mathtt{a} \oslash 2^k = \mathtt{a}/2^k$ jetzt durchaus in den denormalisierten Zahlenbereich $U = (-\mathtt{MinReal}, +\mathtt{MinReal})$ fallen können. Durch die Bedingung (3.3) des letzten Beispiels wird dies jedoch verhindert, da dort das echte Intervall $\widetilde{A} = [4.296875 \cdot 10^{-2}, 10]$ neben $\mathtt{a} = 4.296875 \cdot 10^{-2}$ noch weitere Maschinenzahlen enthält, die ebenfalls rundungsfehlerfrei durch 2^k zu dividieren sind!

Kommen wir jetzt zur rundungsfehlerfreien **Multiplikation**. In Satz 3.1.3 beziehen sich die beiden ersten Bedingungen auf die Multiplikation mit dem Faktor 2^k, so dass diese Fälle auf die Division durch 2^{-k} zurückgeführt werden können. Da Beispiele für solche Divisionen schon behandelt wurden, bleibt jetzt nur noch die dritte Bedingung in Satz 3.1.3 auf Seite 51. Im nächsten Beispiel zeigen wir zunächst, dass diese dritte Bedingung für eine rundungsfehlerfreie Multiplikation nicht notwendig ist.

Beispiel 4 (Multiplikation) Wir wählen $\widetilde{a} = \widetilde{b} = 1+1\cdot2^{-26}$ und erhalten damit $\widetilde{a} \cdot \widetilde{b} = (1+1\cdot2^{-26})^2 = \mathbf{1} + 1\cdot2^{-25} + \mathit{1}\cdot2^{-52} = \mathbf{1}.00\ldots00100\ldots001_{bin} \cdot 2^0$, wobei die 52. Nachkommaziffer *1* den Wert $\mathit{1}\cdot2^{-52}$ darstellt, so dass die rundungsfehlerfreie Multiplikation wegen $\widetilde{a} \cdot \widetilde{b} = \widetilde{a} \odot \widetilde{b} \in S(2,53)$ garantiert ist. Wegen der Darstellung

$$\widetilde{a} = \mathbf{1} + 1 \cdot 2^{-26} = \mathbf{1}.00\ldots001\underbrace{00\ldots00}_{\text{26 Ziffern 0}}{}_{bin} \cdot 2^0$$

gilt: $z_{lead}(\widetilde{a}) = 0,\ z_{trail}(\widetilde{a}) = 26$, und die 3. Bedingung in Satz 3.1.3 lautet damit:

$$2 \cdot (z_{lead}(\widetilde{a}) + z_{trail}(\widetilde{a})) = 2 \cdot (0 + 26) = 52 \geq 53$$

Die 3. Bedingung des Satzes 3.1.3 ist also nicht erfüllt, und somit ist diese Bedingung für die rundungsfehlerfreie Multiplikation nur hinreichend aber nicht notwendig!

Beispiel 5 (Multiplikation) Mit $\widetilde{a} = 1 + 2^{-1} + 2^{-26}$, $\widetilde{b} = 1 + 2^{-1} + 2^{-25}$ erhält man $\widetilde{a} \cdot \widetilde{b} = (1 + 2^{-3} + 2^{-25} + 2^{-28} + 2^{-52}) \cdot 2^1$, womit die 53-stellige normierte Produktmantisse

$$\begin{aligned} & (1 + 2^{-3} + 2^{-25} + 2^{-28} + 2^{-52}) \\ = \; & \underbrace{1.00100000000000000000000100100000000000000000000001}_{53-stellige\ binäre\ Mantisse\ im\ double-Format}{}_{bin} \end{aligned}$$

im double–Format exakt darstellbar ist, so dass die Multiplikation $\widetilde{a} \cdot \widetilde{b} = \widetilde{a} \odot \widetilde{b}$ auf der Maschine rundungsfehlerfrei durchführbar sein muss. Mit Hilfe der 3. Bedingung des Satzes 3.1.3 lässt sich dies nun leicht bestätigen, denn mit $z_{lead}(\widetilde{a}) = 0,\ z_{trail}(\widetilde{a}) = 26$ und $z_{lead}(\widetilde{b}) = 0,\ z_{trail}(\widetilde{a}) = 27$ erhält man:

$$z_{lead}(\widetilde{a}) + z_{trail}(\widetilde{a}) + z_{lead}(\widetilde{b}) + z_{trail}(\widetilde{b}) = 0 + 26 + 0 + 27 = 53 \geq 53,$$

womit die rundungsfehlerfreie Multiplikation $\widetilde{a} \cdot \widetilde{b} = \widetilde{a} \odot \widetilde{b}$ nach Satz 3.1.3 durch diese dritte Bedingung auch tatsächlich garantiert wird.

Für die rundungsfehlerfreie **Addition** betrachten wir nach Satz 3.1.1 von Seite 50 zunächst nur die erste Bedingung. Die beiden Forderungen $\mathrm{Inf}(\widetilde{A}/\widetilde{B}) \geq -2$ und $\mathrm{Sup}(\widetilde{A}/\widetilde{B}) \leq -\frac{1}{2}$ verlangen $0 \notin \widetilde{B}$ und auch $0 \notin \widetilde{A}$, so dass die Maschinenzahlen $\widetilde{a} \in \widetilde{A}$ und $\widetilde{b} \in \widetilde{B}$ in den Operandenintervallen $\widetilde{A}, \widetilde{B}$ jeweils gleiche Vorzeichen haben und die Produkte $\widetilde{a} \cdot \widetilde{b}$ stets negativ sein müssen. Die 1. Bedingung in Satz 3.1.1 ist damit eine Bedingung für die rundungsfehlerfrei Addition positiver und negativer Maschinenzahlen.

Beispiel 6 (Addition) Wir wählen $\widetilde{A} = [2, 6]$ und $\widetilde{B} = [-4, -3]$, und wegen $\widetilde{A}/\widetilde{B} = [-2, -\frac{1}{2}]$ ist damit die erste Bedingung von Satz 3.1.1 erfüllt, so dass gilt: $\widetilde{a} \oplus \widetilde{b} = \widetilde{a} + \widetilde{b}$ für alle $\widetilde{a} \in \widetilde{A} \cap S(2, 53)$ und für alle $\widetilde{b} \in \widetilde{B} \cap S(2, 53)$, d.h. die Addition der Maschinenzahlen erfolgt rundungsfehlerfrei.

Beispiel 7 (Addition) Wir wählen $\widetilde{A} = [2, 6]$ und $\widetilde{B} = [-5, -3]$. Wegen $\widetilde{A}/\widetilde{B} = [-2, -0.4]$ ist jetzt die erste Bedingung von Satz 3.1.1 nicht erfüllt. Trotzdem gilt: $\widetilde{a} \oplus \widetilde{b} = \widetilde{a} + \widetilde{b}$ für alle $\widetilde{a} \in \widetilde{A} \cap S(2, 53)$ und für alle $\widetilde{b} \in \widetilde{B} \cap S(2, 53)$, d.h. die Addition der Maschinenzahlen erfolgt auch jetzt rundungsfehlerfrei, d.h. die 1. Bedingung in Satz 3.1.1 ist für die rundungsfehlerfreie Addition nur hinreichend aber nicht notwendig!

Beispiel 8 (Addition) Wir wählen $\widetilde{A} = [2, 6]$ und $\widetilde{B} = [-7, -3]$. Wegen $\widetilde{A}/\widetilde{B} = [-2, -2/7]$ ist die erste Bedingung von Satz 3.1.1 wieder nicht erfüllt. Die Addition $\widetilde{a} + \widetilde{b}$ ist aber jetzt für alle $\widetilde{a} \in \widetilde{A} \cap S(2, 53)$ und für alle $\widetilde{b} \in \widetilde{B} \cap S(2, 53)$ nicht mehr rundungsfehlerfrei, denn mit $\widetilde{a} = 1.00\ldots001 \cdot 2^1$ und $\widetilde{b} = -1.100\ldots001 \cdot 2^2$ gilt: $\widetilde{a} + \widetilde{b} = (1 + 2^{-52}) \cdot 2^1 - (1 + 2^{-1} + 2^{-52}) \cdot 2^2 = -(1 + 2^{-53}) \cdot 2^2$; wegen 2^{-53} ist damit die Mantisse $(1 + 2^{-53})$ im **IEEE** double-Format nicht mehr darstellbar, und man erhält: $\widetilde{a} + \widetilde{b} \neq \widetilde{a} \oplus \widetilde{b}$.

In Satz 3.1.1 betrachten wir jetzt für die rundungsfehlerfreie Addition die zweite Bedingung $e(|\widetilde{A}+\widetilde{B}|) \leq \min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\}$, wobei die Funktion $e_{IS}(X)$ auf Seite 50 definiert ist. In den folgenden Beispielen betrachten wir zunächst nur Punktintervalle $\widetilde{A} = [\widetilde{a}, \widetilde{a}],\ \widetilde{B} = [\widetilde{b}, \widetilde{b}]$.

Beispiel 9 (Addition) Wir wählen $\widetilde{A} = [\widetilde{a}, \widetilde{a}],\ \widetilde{B} = [\widetilde{b}, \widetilde{b}]$ mit $\widetilde{a} = (1+2^{-52}) \cdot 2^0 = 1.00\ldots001 \cdot 2^0$ und $\widetilde{b} = 1 = 1.00\ldots00 \cdot 2^0$. Wegen $\widetilde{a} + \widetilde{b} = (1 + 2^{-53}) \cdot 2^1$ ist die normalisierte Mantisse $1 + 2^{-53}$ mit ihren 54 Binärziffern im **IEEE** double-Format mit nur 53 Mantissenziffern nicht darstellbar, d.h. es gilt: $\widetilde{a}+\widetilde{b} \neq \widetilde{a}\oplus\widetilde{b}$. Die Addition $\widetilde{a} \oplus \widetilde{b}$ ist auf der Maschine also nicht rundungsfehlerfrei, so dass die 2. Bedingung von Satz 3.1.1 nicht erfüllt sein kann. Zum Nachweis berechnen wir $e(|\widetilde{A} + \widetilde{B}|) = e(\widetilde{a} + \widetilde{b}) = 1$ und $\min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\} = \min\{e(\widetilde{a}) + z_{trail}(\widetilde{a}), e(\widetilde{b}) + z_{trail}(\widetilde{b})\} = \min\{0 + 0, 0 + 52\} = 0$, d.h. die zweite Bedingung von Satz 3.1.1 ist wie erwartet nicht erfüllt.

Im obigen Beispiel kann die Addition nicht rundungsfehlerfrei durchgeführt werden, weil bei vollbesetzter Mantisse von $\widetilde{a}$, d.h. $z_{trail}(\widetilde{a}) = 0$, die Addition einen Übertrag erzeugt und dadurch die exakte Mantisse der Summe für das **IEEE** double-Format zu lang wird. Im Gegensatz dazu ist die Addition im nächsten Beispiel rundungsfehlerfrei, da die exakte Mantisse der Summe gerade noch mit 53 Mantissenziffern dargestellt werden kann.

Beispiel 10 (Addition) Wir wählen $\widetilde{A} = [\widetilde{a}, \widetilde{a}],\ \widetilde{B} = [\widetilde{b}, \widetilde{b}]$ mit $\widetilde{a} = 1 \cdot 2^0 = 1.00\ldots00 \cdot 2^0$ und $\widetilde{b} = 1.00\ldots00 \cdot 2^{-52}$. Wegen $\widetilde{a} + \widetilde{b} = (1 + 2^{-52}) \cdot 2^0$ ist die normalisierte Mantisse $1 + 2^{-52}$ mit ihren 53 Binärziffern im **IEEE** double-Format exakt darstellbar, d.h. es gilt: $\widetilde{a} + \widetilde{b} = \widetilde{a} \oplus \widetilde{b}$. Die Addition $\widetilde{a} \oplus \widetilde{b}$ erfolgt auf der Maschine also rundungsfehlerfrei. Zum Nachweis der 2. Bedingung von Satz 3.1.1 gilt: $e(|\widetilde{A}+\widetilde{B}|) = e(\widetilde{a}+\widetilde{b}) = 0$ und $\min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\} = \min\{e(\widetilde{a})+z_{trail}(\widetilde{a}), e(\widetilde{b})+z_{trail}(\widetilde{b})\} = \min\{0 + 52, -52 + 52\} = 0$, d.h. die zweite Bedingung von Satz 3.1.1 ist erfüllt.

Beispiel 11 (Addition) Wir wählen $\widetilde{A} = [\widetilde{a}, \widetilde{a}]$, $\widetilde{B} = [\widetilde{b}, \widetilde{b}]$ mit $\widetilde{a} = 1 \cdot 2^0 = 1.00\ldots00 \cdot 2^0$ und $\widetilde{b} = 1.100\ldots00 \cdot 2^{-52}$. Wegen $\widetilde{a} + \widetilde{b} = (1 + 2^{-52} + 2^{-53}) \cdot 2^0$ ist die normalisierte Mantisse $1 + 2^{-52} + 2^{-53}$ mit ihren 54 Binärziffern im **IEEE** double-Format nicht mehr exakt darstellbar, d.h. es gilt: $\widetilde{a} + \widetilde{b} \neq \widetilde{a} \oplus \widetilde{b}$. Die Addition $\widetilde{a} \oplus \widetilde{b}$ erfolgt auf der Maschine also nicht rundungsfehlerfrei. Zum Nachweis, dass die 2. Bedingung von Satz 3.1.1 jetzt nicht erfüllt sein kann, berechnen wir: $e(|\widetilde{A} + \widetilde{B}|) = e(\widetilde{a} + \widetilde{b}) = 0$ und $\min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\} = \min\{e(\widetilde{a}) + z_{trail}(\widetilde{a}), e(\widetilde{b}) + z_{trail}(\widetilde{b})\} = \min\{0 + 52, -52 + 51\} = -1$, d.h. die zweite Bedingung von Satz 3.1.1 ist in der Tat nicht erfüllt.

Im abschließenden Beispiel geben wir zwei echte Intervalle $\widetilde{A}, \widetilde{B}$ an, für die im Satz 3.1.1 die 2. Bedingung erfüllt wird.

Beispiel 12 (Addition) Wir wählen $\widetilde{A} = [2, 2.25]$, $\widetilde{B} = [-1.5, -1]$. Zum Nachweis der 2. Bedingung von Satz 3.1.1 berechnen wir: $e(|\widetilde{A} + \widetilde{B}|) = e(|[0.5, 1.25]|) = e(1.25) = 0$ und $\min\{e_{IS}(\widetilde{A}), e_{IS}(\widetilde{B})\} = \min\{e(\langle\widetilde{A}\rangle), e(\langle\widetilde{B}\rangle)\} = \min\{e(2), e(1)\} = \min\{1, 0\} = 0$. Damit ist die zweite Bedingung von Satz 3.1.1 erfüllt, und für alle $\widetilde{a} \in \widetilde{A} \cap S(2, 53)$ und für alle $\widetilde{b} \in \widetilde{B} \cap S(2, 53)$ gilt: $\widetilde{a} + \widetilde{b} = \widetilde{a} \oplus \widetilde{b}$, d.h. die Addition ist in diesem Beispiel rundungsfehlerfrei durchführbar. Beachten Sie bitte, dass wegen $\widetilde{A}/\widetilde{B} = [-2.25, -4/3]$ die erste Bedingung von Satz 3.1.1 jetzt nicht erfüllt ist und die rundungsfehlerfrei Addition nur durch die zweite Bedingung garantiert wird.

Anmerkungen:

- Beachten Sie bitte, dass bei der automatischen Fehlerabschätzung die Sätze 3.1.1 bis 3.1.4 in den entsprechenden Hilfsfunktionen der Tabelle 2.4 auf Seite 40 schon eingearbeitet sind, so dass sich der Anwender um die korrekte Handhabung dieser vier Sätze nicht zu kümmern braucht!

- Im Beispiel 4 haben wir mit $\widetilde{a} = \widetilde{b} = 1 + 2^{-26}$ einen Fall kennen gelernt, bei dem die Multiplikation $\widetilde{a} \odot \widetilde{b}$ auf der Maschine rundungsfehlerfrei erfolgt, dies aber durch den Satz 3.1.3 nicht erkannt wird. Dem Anwender sollte daher stets bewusst sein, dass bei der automatischen Fehlerabschätzung die berechneten Fehlerschranken nicht immer optimal ausfallen können, und dass man daher in Spezialfällen die Fehlerabschätzung selbst weiter verbessern muss!

3.2 Spezielle Operanden

Schon auf Seite 56 haben wir in der letzten Anmerkung darauf hingewiesen, dass die Sätze 3.1.1 bis 3.1.4 nicht alle Fälle abdecken, in denen die Grundoperationen auf der Maschine rundungsfehlerfrei durchgeführt werden können. Es gibt jedoch auch Fälle, in denen z.B. der tatsächliche absolute Fehler sehr viel kleiner ist als die mit den Funktionen aus den Modulen `abs_relm` oder `abs_relh` berechneten Fehlerschranken. Als Beispiel betrachten wir die Addition der Maschinenzahlen $\widetilde{a} = 1$ und $\widetilde{x} = 2^{+300}$. Für den absoluten Fehler gilt bei maximal genauer Arithmetik: $|(1 + \widetilde{x}) - (1 \oplus \widetilde{x})| = |(1 + \widetilde{x}) - \widetilde{x}| = 1$. Nach (2.16) und (2.19) berechnet sich die Oberschranke des absoluten Fehlers wegen $\Delta_{dat}(+) = 0$ zu: $\overline{\varepsilon}(0+|1+[2^{300}, 2^{300}]|) = 2^{-53} \cdot (1+2^{300}) \gg 1$. Damit liefert die Funktion `abs_addm1()` aus dem Modul `abs_relm` eine Überschätzung des tatsächlichen absoluten Fehlers 1 um ganze 74 Zehnerpotenzen! In den folgenden Unterabschnitten werden wir daher für öfters auftretende spezielle Terme absolute und relative Fehlerschranken angeben, wenn die entsprechenden Grundoperationen in hochgenauer Arithmetik ausgeführt werden. Die Betrachtung einer maximal genauen Arithmetik hatten wir schon auf Seite 35 ausgeschlossen.

3.2.1 $T(x) = 1 + x$

Für alle reellen $x \in \mathtt{X} \in IR$ ist der absolute und falls möglich auch der relative Fehler abzuschätzen, wenn $T(x) = 1 + x$ auf der Maschine ausgewertet wird. Wir interpretieren dabei das Maschinenintervall `X` wieder als Wertebereich einer Funktion deren Funktionswerte $x \in I\!R$ auf der Maschine als Maschinenzahlen $\widetilde{x} \in S(2, 53)$ fehlerhaft berechnet wurden. Die Fehlerschranke $\Delta(x)$ des absoluten Fehlers ist für die gestörten Funktionswerte $\widetilde{x}$ definiert durch

$$|x - \widetilde{x}| \leq \Delta(x) \quad \text{für alle } x \in \mathtt{X} \in IR \tag{3.5}$$

Die gestörten Maschinenzahlen $\widetilde{x} \in S(2, 53)$ werden dann durch das Maschinenintervall $\widetilde{\mathtt{X}} := \mathtt{X} \oplus [-\Delta(x), +\Delta(x)]$ eingeschlossen, wobei $\oplus$ hier die in **C-XSC** definierte Intervalladdition bedeutet[1]. Im Falle $0 \notin \mathtt{X}$ kann die relative Fehlerschranke $\varepsilon(x)$ definiert werden durch

$$\left| \frac{x - \widetilde{x}}{x} \right| \leq \varepsilon(x) \quad \text{für alle } x \in \mathtt{X} \in IR \tag{3.6}$$

und eine Oberschranke des absoluten Fehlers lautet jetzt: $\Delta(x) := \varepsilon(x) \cdot |\mathtt{X}|$. Die gestörten Maschinenzahlen $\widetilde{x} \in S(2, 53)$ werden dann durch das Maschinenintervall

[1] $\Delta(x) \in I\!R$ ist dabei durch eine Maschinenzahl `delx` $\in S(2, 53)$ nach oben abzuschätzen.

$\widetilde{\mathtt{X}} := \mathtt{X} \oplus [-\Delta(x), +\Delta(x)]$ eingeschlossen[2]. Für die Abschätzung des absoluten Fehlers gilt dann:

$$\begin{aligned} |T(x) - \widetilde{T}(\widetilde{x})| &= |(T(x) - T(\widetilde{x})) + (T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))| \\ &\leq |T(x) - T(\widetilde{x})| + |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \\ &= \Delta_{dat}(T) + \Delta_{rnd}(T) \quad \text{für alle } x \in \mathtt{X} \end{aligned} \tag{3.7}$$

Ist für die gestörten Werte $\widetilde{x}$ eine absolute Fehlerschranke $\Delta(x)$ gegeben, so gilt für den fortgepflanzten Datenfehler:

$$\Delta_{dat}(T) := |T(x) - T(\widetilde{x})| = |1 + x - (1 + \widetilde{x})| = |x - \widetilde{x}| \leq \Delta(x), \quad x \in \mathtt{X} \tag{3.8}$$

Ist für die gestörten Werte $\widetilde{x}$ eine relative Fehlerschranke $\varepsilon(x)$ gegeben, so gilt für den fortgepflanzten Datenfehler:

$$\Delta_{dat}(T) := |T(x) - T(\widetilde{x})| = \left| \frac{x - \widetilde{x}}{x} \right| \cdot |x| \leq \varepsilon(x) \cdot |X|, \quad x \in \mathtt{X} \tag{3.9}$$

Für alle Maschinenzahlen $\widetilde{x} \in \widetilde{\mathtt{X}}$ ist der Rundungsfehler definiert durch:

$$\Delta_{rnd}(T) := |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| = |(1 + \widetilde{x}) - (1 \oplus \widetilde{x})|, \quad \widetilde{x} \in \widetilde{\mathtt{X}} \tag{3.10}$$

Für die einzelnen Bereiche der Maschinenzahlen $\widetilde{x} \in S(2, 53)$ sind die in (3.10) definierten Rundungsfehler $\Delta_{rnd}(T)$ in nachfolgender Tabelle zusammengestellt:

[2] Im Programmtext werden wir $\widetilde{\mathtt{X}}$ mit `Xt` bezeichnen.

$\widetilde{x} \in X$	$\Delta_{rnd}(T)$	$\widetilde{x} \in$	$\Delta_{rnd}(T)$
$[-\texttt{MaxReal}, -2^{54})$	$\varepsilon(h) \cdot \lvert 1 + \widetilde{x}\rvert$	$[2, 3]$	0
$[-2^{54}, -2^{53})$	1	$(3, 4)$	2^{2-53}
$[-2^{53}, -0.5]$	0	$[4, 7]$	0
$(-0.5, -0.25]$	2^{-54}	$(7, 8)$	2^{3-53}
$(-0.25, -0.125]$	$1.5 \cdot 2^{-54}$	$[8, 15]$	0
$(-0.125, -2^{-4}]$	$1.75 \cdot 2^{-54}$	$(15, 16)$	2^{4-53}
$[-2^{-4}, 0]$	2^{-53}	$[2^{53}, \texttt{pred}(2^{54})]$	$2^{\texttt{expo}(\widetilde{x})-53} - 1 = 1$
$[0, 0.5]$	2^{-52}	$[2^{54}, \texttt{pred}(2^{55})]$	$2^{\texttt{expo}(\widetilde{x})-53} - 1 = 3$
$[0.5, 1)$	2^{-53}	$[2^{55}, \texttt{pred}(2^{56})]$	$2^{\texttt{expo}(\widetilde{x})-53} - 1 = 7$
$[1, 2)$	2^{-52}	$[2^{56}, \texttt{pred}(2^{57})]$	$2^{\texttt{expo}(\widetilde{x})-53} - 1 = 15$

Tabelle 3.1: Rundungsfehler $\Delta_{rnd}(T)$ für alle $\widetilde{x} \in S(2, 53)$

Wir wollen zuerst die in Tabelle 3.1 zusammengestellten absoluten Fehlerschranken $\Delta_{rnd}(T)$ mit den entsprechenden Schranken `abs` vergleichen, die wir ja schon mit Hilfe der Funktion `abs_addh1(interval(1),0,X,0,R,abs)` berechnen können. Die Schranke $\Delta_{rnd}(T) = \varepsilon(h) \cdot |1 + \widetilde{x}|$ ergibt sich für den Bereich $[-\texttt{MaxReal}, -2^{+54}]$ direkt aus der Definition des relativen Fehlers ε bei hochgenauer Arithmetik:

$$\widetilde{T}(\widetilde{x}) - T(\widetilde{x}) = T(\widetilde{x}) \cdot \varepsilon, \quad |\varepsilon| \leq \varepsilon(h) := 2^{-25} \quad \rightsquigarrow$$
$$|\widetilde{T}(\widetilde{x}) - T(\widetilde{x})| \leq \Delta_{rnd}(T) := \varepsilon(h) \cdot |1 + \widetilde{x}|,$$

und auch die Funktion `abs_addh1(...)` berechnet ihren Rückgabewert `abs` mit dem gleichen Produkt $\varepsilon(h) \cdot |1 + \widetilde{x}|$. Für die restlichen Bereiche sind die Fehlerschranken $\Delta_{rnd}(T)$ und `abs` zum Vergleich in folgender Tabelle zusammengestellt:

Man erkennt, dass in fast allen Bereichen die absoluten Fehlerschranken $\Delta_{rnd}(T)$ mindestens um den Faktor 2 kleiner ausfallen als die Schranken `abs`, so dass wir bei zukünftigen Fehlerabschätzungen natürlich die verbesserten $\Delta_{rnd}(T)$-Schranken benutzen werden, vergleichen Sie dazu die Bemerkungen ab Seite 69.

Bevor wir die Richtigkeit der Schranken $\Delta_{rnd}(T)$ nachweisen, wird ein Programm vorgestellt, mit dem man diese Schranken wenigstens an den Rändern der Intervalle X bestätigen kann:

```
// Programm:  Schranken.cpp;
// Test der absoluten Fehlerschranken;
// x aus X = [2^(54),pred(2^(55))]
```

$\widetilde{x} \in X$	$\Delta_{rnd}(T)$	`abs`
$[-2^{54}, -2^{53})$	1.000000	4.000000
$[-2^{53}, -0.5]$	0.000000	2.000000
$(-0.5, -0.25]$	$5.551116 \cdot 10^{-17}$	$1.665335 \cdot 10^{-16}$
$(-0.25, -0.125]$	$8.326673 \cdot 10^{-17}$	$1.942891 \cdot 10^{-16}$
$(-0.125, -2^{-4}]$	$9.714452 \cdot 10^{-17}$	$2.081669 \cdot 10^{-16}$
$[-2^{-4}, 0]$	$1.110224 \cdot 10^{-16}$	$2.220447 \cdot 10^{-16}$
$[0, 0.5]$	$2.220447 \cdot 10^{-16}$	$3.330670 \cdot 10^{-16}$
$[0.5, 1)$	$1.110224 \cdot 10^{-16}$	$4.440893 \cdot 10^{-16}$
$[1, 2)$	$2.220447 \cdot 10^{-16}$	$6.661339 \cdot 10^{-16}$
$[2, 3]$	0.000000	$8.881785 \cdot 10^{-16}$
$(3, 4)$	$4.440893 \cdot 10^{-16}$	$1.110224 \cdot 10^{-15}$
$[4, 7]$	0.000000	$1.776357 \cdot 10^{-15}$
$(7, 8)$	$8.881785 \cdot 10^{-16}$	$1.998402 \cdot 10^{-15}$
$[8, 15]$	0.000000	$3.552714 \cdot 10^{-15}$
$(15, 16)$	$1.776357 \cdot 10^{-15}$	$3.774759 \cdot 10^{-15}$
$[2^{53}, \texttt{pred}(2^{54})]$	1.000000	4.000000
$[2^{54}, \texttt{pred}(2^{55})]$	3.000000	8.000000
$[2^{55}, \texttt{pred}(2^{56})]$	7.000000	16.000000

Tabelle 3.2: Vergleich der Schranken $\Delta_{rnd}(T)$ und `abs`

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

using namespace cxsc;
using namespace std;

real absF(const real& x)
{
    interval x2,R;
    real msu,msd,d,del;
    dotprecision dot,dot1;
```

```
    dot = 1.0;
    dot = dot + x; // dot = 1+x;
    msu = addu(1.0,x);
    msd = addd(1.0,x);
    dot1 = dot;  // dot1 = dot = 1+x;
    dot = dot - msu; // dot: absoluter Fehler beim Aufrunden
    if (sign(dot)<0) {
        rnd(dot,d,RND_DOWN);
        d = abs(d);}
    else rnd(dot,d,RND_UP);
    dot1 = dot1 - msd; // dot: absoluter Fehler beim Abrunden
    if (sign(dot1)<0) {
        rnd(dot1,del,RND_DOWN);
        del = abs(del);}
    else rnd(dot1,del,RND_UP);
    if(d>del) del = d; // Maximumbildung

    return del;
}
int main()
{
    const real c = 3.0; // Zu testende abs. Fehlerschranke
    real a(comp(0.5,55)), b(pred(comp(0.5,56))); // X = [a,b]
    real del, x=a;

    do {
        del = absF(x);
        if (del>c) {
            cout << "!!!!  del > c  !!!!" << endl;
            cout << SetPrecision(20,20) << "x = " << x << endl;
            cout << "del = " << del << endl;
            exit(1);
        };
        x = succ(x); }
    while (del<=c && x<=b);
}
```

Das obige Programm `Schranken.cpp` überprüft die Richtigkeit der Fehlerschranke $\Delta_{rnd}(T) = 3$ für das Intervall $X = [2^{54}, \texttt{pred}(2^{55})]$ bez. aller Rasterzahlen, beginnend bei $a = 2^{54}$. Sollte eine Schranke größer als 3 ausfallen, so erfolgt die Meldung

```
!!!!  del > c  !!!!
```

mit Programmabbruch. Da aus Laufzeitgründen nicht alle Rasterzahlen des Intervalls X überprüft werden können, muss die Richtigkeit der Fehlerschranken $\Delta_{rnd}(T)$ aus Tabelle 3.2 für jedes Intervall X einzeln nachgewiesen werden:

$X = [-2^{54}, -2^{53})$

Da der absolute Rundungsfehler bei der Maschinenauswertung von $1 \oplus \widetilde{x}$ für alle $\widetilde{x} \in X$ betragsmäßig identisch ist mit dem absolute Rundungsfehler bei Auswertung von $\widetilde{x} \ominus 1$ für alle $\widetilde{x} \in -X = (2^{53}, 2^{54}]$, wählen wir für den folgenden Beweis diese Differenzschreibweise, da die benötigten Bit-Muster hiermit einfacher zu formulieren sind. Für $\widetilde{x} = \mathtt{succ}(2^{53})$, $\widetilde{x} = \mathtt{succ}(\mathtt{succ}((2^{53})))$ und $\widetilde{x} = 2^{54}$ schreiben wir die exakten Differenzen $\widetilde{x} \ominus 1$ als Bit-Muster und können daraus den maximalen Rundungsfehler direkt ablesen:

$$\begin{array}{rcl} \mathtt{succ}(2^{53}) & = & 2^{53} \cdot 1.\overbrace{0000\ldots0001}^{52\text{ Dualziffern}}000 \\ 1 & = & \underline{\qquad\qquad\qquad\qquad 100} \\ \mathtt{succ}(2^{53}) - 1 & = & 2^{53} \cdot 1.0000\ldots0000\,\mathbf{100} \end{array}$$

$$\begin{array}{rcl} \mathtt{succ}(\mathtt{succ}((2^{53}))) & = & 2^{53} \cdot 1.\overbrace{0000\ldots0010}^{52\text{ Dualziffern}}000 \\ 1 & = & \underline{\qquad\qquad\qquad\qquad 100} \\ \mathtt{succ}(\mathtt{succ}((2^{53}))) - 1 & = & 2^{53} \cdot 1.0000\ldots0001\,\mathbf{100} \end{array}$$

$$\begin{array}{rcl} 2^{54} & = & 2^{54} \cdot 1.\overbrace{0000\ldots0000}^{52\text{ Dualziffern}}000 \\ 1 & = & \underline{\qquad\qquad\qquad\qquad 010} \\ 2^{54} - 1 & = & 2^{54} \cdot 0.1111\ldots1111\,\mathbf{110} = 2^{53} \cdot 1.1111\ldots1111\,\mathbf{10} \end{array}$$

Die Bit-Muster aller Differenzen $\widetilde{x} - 1$ haben damit für $z \in \{0, 1\}$ folgende Struktur:

$$\widetilde{x} - 1 = 2^{53} \cdot 1.\overbrace{zzzz\ldots zzzz}^{52\text{ Dualziffern}}\mathbf{100}\ldots \quad \forall \widetilde{x} \in -X = (2^{53}, 2^{54}];$$

Die exakten Differnzen $\widetilde{x} - 1$ liegen damit stets in der Mitte zwischen zwei benachbarten Rasterzahlen, so dass eine **optimale Schranke** des absoluten Rundungsfehlers durch folgende Differenz gegeben ist[3]:

$$\begin{array}{rcl} & & 2^{53} \cdot 1.\overbrace{zzzz\ldots zzzz}^{52\text{ Dualziffern}}\mathbf{100} \\ & - & \underline{2^{53} \cdot 1.zzzz\ldots zzzz\,000} \\ \Delta_{rnd}(T) & = & 2^{53} \cdot 0.\,0000\ \ldots\ 0000\ \mathbf{100} = 2^{53} \cdot 2^{-53} = 1 \end{array}$$

[3]Da $\widetilde{x} - 1$ stets in der Mitte zwischen zwei benachbarten Rasterzahlen liegt, ist $\Delta_{rnd}(T) = 1$ auch eine Schranke des absoluten Rundungsfehlers bei maximal genauer Arithmetik!

Wir haben den Beweis bez. $\Delta_{rnd}(T) = 1$ für das Intervall $X = [-2^{54}, -2^{53})$ ausführlich beschrieben, da die Beweise in den anderen Intervallen ganz ähnlich verlaufen.

$X = [-2^{53}, -0.5]$

Um nachzuweisen, dass in diesem Intervall $\Delta_{rnd}(T) = 0$ eine korrekte und damit auch optimale Fehlerschranke ist, ist nur zu zeigen, dass die maximale Mantissenbreite der exakten Werte $1 + \widetilde{x}$ durch 53 Bits gegeben ist. Für $\widetilde{x} = -0.5$, $\texttt{pred}(-0.5)$, ... bis $\widetilde{x} = -1$ ist dies evident. Für $2^{53} - 1.0$ geben wir das entsprechende Bitmuster an:

$$\begin{array}{rcl} & & \overbrace{}^{52\text{ Dualziffern}} \\ 2^{53} & = & 2^{53} \cdot 1.\overbrace{0000\ldots0000}\,000 \\ 1 & = & \underline{\qquad\qquad\qquad\qquad 100} \\ 2^{53} - 1 & = & 2^{53} \cdot 0.1111\ldots1111\,\mathbf{100} \end{array}$$

Die obige Mantisse besteht damit aus genau 53 Dualziffern 1, d.h. die Differenz ist in $S(2, 53)$ darstellbar. Diese gleiche maximale Mantissenbreite erhält man nur noch bei der Berechnung von $1 - \texttt{succ}(0.5)$.

$X = (-1/8, -1/16]$
Wir zeigen zunächst, dass der maximale Rundungsfehler $\Delta_{rnd}(T) = 1.75 \cdot 2^{-54}$ für die Differenz $1 - \texttt{pred}(1/8)$ angenommen wird:

$$\begin{aligned} 1 &= 1.\overbrace{000000\ldots0000}^{52\text{ Dualziffern}}000000\ldots \\ \texttt{pred}(1/8) &= \underline{0.000111\ldots1111\,111100}\ldots \\ 1-\texttt{pred}(1/8) &= 0.111000\ldots0000\,000100\ldots \end{aligned}$$

Die exakte Differenz liegt also dicht an der linken Rasterzahl, so dass bei hochgenauer Arithmetik der Rundungsfehler durch den Abstand zur rechten Rasterzahl gegeben ist:

$$\begin{aligned} \text{rechte Rasterzahl} &= 0.\overbrace{111000\ldots0001}^{53\text{ Dualziffern}}000000\ldots \\ 1-\texttt{pred}(1/8) &= \underline{0.111000\ldots0000\,001000}\ldots \\ \text{rechte Rasterzahl} - (1-\texttt{pred}(1/8)) &= 0.000000\ldots0000\,111000\ldots \\ &= 2^{-54}+2^{-55}+2^{-56} = 1.75\cdot 2^{-54}; \end{aligned}$$

Wir geben jetzt noch das Bitmuster der Differenz $1 - \texttt{succ}(1/16)$ an:

$$\begin{aligned} 1 &= 1.\overbrace{000000\ldots0000}^{52\text{ Dualziffern}}000000\ldots \\ \texttt{succ}(1/16) &= \underline{0.000100\ldots0000\,000100}\ldots \\ 1-\texttt{succ}(1/16) &= 0.111011\ldots1111\,1\mathbf{111}00\ldots \end{aligned}$$

und der maximale Abstand zur linken Rasterzahl ist gegeben durch das Bitmuster:

$$0.000000\ldots0000\,0\mathbf{111}00\ldots = 2^{-54}+2^{-55}+2^{-56} = 1.75\cdot 2^{-54};$$

Folgende Bitmuster für $-\widetilde{x}$ liefern damit den maximalen absoluten Rundungsfehler $\Delta_{rnd}(T) = 1.75 \cdot 2^{-54}$:

$$-\widetilde{x} = 0.0001\overbrace{zzzzz\ldots zzzzz}^{49\text{ Ziffern }z\in\{0,1\}}\mathbf{111}000\ldots \tag{3.11}$$

$$-\widetilde{x} = 0.0001\overbrace{zzzzz\ldots zzzzz}^{49\text{ Ziffern }z\in\{0,1\}}\mathbf{001}000\ldots \tag{3.12}$$

$X_1 = (-0.5, -0.25]$, $X_2 = (-0.25, -0.125]$
Für diese Intervalle kann man zu (3.11) und (3.12) analoge Bitmuster angeben, mit denen man die jeweiligen maximalen Fehlerschranken direkt bestimmen kann.

Für $X_2 = (-0.25, -0.125]$ lauten diese Bitmuster:

$$\begin{aligned} -\widetilde{x} &= 0.001\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern } z\in\{0,1\}}\mathbf{11}000\ldots \\ -\widetilde{x} &= 0.001\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern } z\in\{0,1\}}\mathbf{01}000\ldots \end{aligned}$$

Aus beiden Bitmustern ergibt sich ganz analog zum Intervall $X = (-1/8, -1/16]$ die Schranke für den absoluten Fehler $\Delta_{rnd}(T) = 1.5 \cdot 2^{-54}$.
Für $X_1 = (-0.5, -0.25]$ reduzieren sich die beiden Bitmuster auf:

$$-\widetilde{x} = 0.01\overbrace{zzzzz\ldots zzzzz}^{51\text{ Ziffern } z\in\{0,1\}}\mathbf{1}000\ldots$$

und daraus erhält man direkt die optimale absolute Schranke: $\Delta_{rnd}(T) = \mathbf{1} \cdot 2^{-54}$.

$X = (-1/16, 0)$
Für das Teilintervall $(-1/16, -1/32]$ erhält man mit den Bitmustern

$$\begin{aligned} -\widetilde{x} &= 0.00001\overbrace{zzzzz\ldots zzzzz}^{48\text{ Ziffern } z\in\{0,1\}}\mathbf{1111}000\ldots \\ -\widetilde{x} &= 0.00001\overbrace{zzzzz\ldots zzzzz}^{48\text{ Ziffern } z\in\{0,1\}}\mathbf{0001}000\ldots \end{aligned}$$

die maximale Fehlerschranke $2^{-54} + 2^{-55} + 2^{-56} + 2^{-57} = 1.875 \cdot 2^{-54} < 2^{-53}$.
Für das Teilintervall $(-2^{-51}, -2^{-52}]$ liefern ganz analoge Überlegungen, dass mit den folgenden Bitmustern und $z \in \{0, 1\}$

$$\begin{aligned} -\widetilde{x} &= 0.\overbrace{0000\ldots0000}^{52-1\text{ Ziffern }0}1z\overbrace{1111\ldots1111}^{52-1\text{ Ziffern }1} \\ -\widetilde{x} &= 0.\overbrace{0000\ldots0000}^{52-1\text{ Ziffern }0}1z\overbrace{0000\ldots0001}^{52-1\text{ Ziffern }1} \end{aligned}$$

der Rundungsfehler maximal wird: $2^{-54} + 2^{-55} + \ldots + 2^{-104} < 2^{-53} = \Delta_{rnd}(T)$.
Für das Teilintervall $(-2^{-52}, -2^{-53}]$ liefern die folgenden Bitmuster

$$\begin{aligned} -\widetilde{x} &= 0.\overbrace{0000\ldots0000}^{52\text{ Ziffern }0}\overbrace{1111\ldots1111}^{53\text{ Ziffern }1} \\ -\widetilde{x} &= 0.\overbrace{0000\ldots0000}^{52\text{ Ziffern }0}\overbrace{1000\ldots0001}^{53\text{ Ziffern}} = \texttt{succ}(2^{-53}) \end{aligned}$$

den maximalen Rundungsfehler: $2^{-54} + 2^{-55} + \ldots + 2^{-105} < 2^{-53} = \Delta_{rnd}(T)$.
Für das Teilintervall $(-2^{-53}, -2^{-54}]$ liefert das folgende Bitmuster

$$-\widetilde{x} = 0.\overbrace{0000\ldots0000}^{52\text{ Ziffern }0}0\overbrace{1111\ldots1111}^{53\text{ Ziffern }1} = \texttt{pred}(2^{-53})$$

den maximalen Rundungsfehler: $2^{-54} + 2^{-55} + \ldots + 2^{-106} < 2^{-53} = \Delta_{rnd}(T)$.

Im Teilintervall $(-2^{-54}, -2^{-55}]$ liefert $-\widetilde{x} = 2^{-55}$ den maximalen Rundungsfehler: $2^{-54} + 2^{-55} < 2^{-53} = \Delta_{rnd}(T)$.
Im Teilintervall $(-2^{-55}, -2^{-56}]$ liefert $-\widetilde{x} = 2^{-56}$ den maximalen Rundungsfehler: $2^{-54} + 2^{-55} + 2^{-56} < 2^{-53} = \Delta_{rnd}(T)$, und man zeigt analog, dass in allen nachfolgenden Teilintervallen und auch für $-\widetilde{x} \in [0, \texttt{MinReal}]$ das Maximum des absoluten Rundungsfehlers stets kleiner bleibt als 2^{-53}. Damit haben wir die Fehlerschranken $\Delta_{rnd}(T)$ aus Tabelle 3.1 von Seite 59 für nagetive Maschinenzahlen $\widetilde{x}$ nachgewiesen.

$X = (0, 0.5)$
Aus den Bitmustern für $1 + \widetilde{x}$ und $\widetilde{x} = \texttt{pred}(0.5)$, $\widetilde{x} = \texttt{pred}(\texttt{pred}((0.5))$, usw. erkennt man unmittelbar, dass bei hochgenauer Arithmetik die absoluten Rundungsfehler stets kleiner bleiben als 2^{-52}, aber mit $\widetilde{x} \to 0$ diesen Wert gut approximieren. Beachten Sie bitte, dass bei der vorausgesetzten hochgenauen Arithmetik der Rundungsfehler durch den maximalen Abstand der exakten Summe $1 + \widetilde{x}$ zu den beiden benachbarten Rasterzahl definiert ist, falls $1 + \widetilde{x}$ nicht exakt darstellbar ist.

$X = (0.5, 1)$
Man erkennt unmittelbar, dass mit den Bitmustern

$$0.5 < \widetilde{x} := 0.1\overbrace{zzzzz\ldots zzzzz}^{51\text{ Ziffern } z\in\{0,1\}}1 < 1$$

die exakte Summe $1 + \widetilde{x}$ den maximalen Rundungsfehler $\Delta_{rnd}(T) := 2^{-53}$ liefert, wobei die Summe $1 + \widetilde{x}$ genau in der Mitte ihrer benachbarten Rasterzahlen liegt.

$X = (1, 2)$
Man erkennt auch in diesem Intervall, dass mit den Bitmustern

$$1 < \widetilde{x} := 1.\overbrace{zzzzz\ldots zzzzz}^{51\text{ Ziffern } z\in\{0,1\}}1 < 2$$

die exakte Summe $1 + \widetilde{x}$ den maximalen Rundungsfehler $\Delta_{rnd}(T) := 2^{-52}$ liefert, wobei die Summe $1 + \widetilde{x}$ genau in der Mitte ihrer benachbarten Rasterzahlen liegt.

$X = [2, 3)$
Man erkennt unmittelbar, dass mit den möglichen Bitmustern

$$2 \leq \widetilde{x} := 10.\overbrace{zzzzz\ldots zzzzz}^{51\text{ Ziffern } z\in\{0,1\}} < 3$$

$$11.\overbrace{zzzzz\ldots zzzzz}^{51\text{ Ziffern } z\in\{0,1\}}$$

die exakte Summe $1 + \widetilde{x} =$ mit maximal 53 Mantissen-Bits in $S(2, 53)$ exakt darstellbar ist.

$X = (3, 4)$
Man erkennt unmittelbar, dass mit den Bitmustern

$$3 < \widetilde{x} := 11.\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern }z\in\{0,1\}}1 < 4$$

$$1 + \widetilde{x} = 100.\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern }z\in\{0,1\}}1$$

die exakte Summe genau in der Mitte benachbarter Rasterzahlen liegt und dass der maximale Rundungsfehler durch $\Delta_{rnd}(T) := 2^{-51}$ gegeben ist.

$X = [4, 7)$
Man erkennt unmittelbar, dass mit allen möglichen Bitmustern

$$4 \le \widetilde{x} := 1z\,0.\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern }z\in\{0,1\}}00\ldots < 7$$

die exakte Summe $1+\widetilde{x} = 1z1.\overbrace{zzzzz\ldots zzzzz}^{50\text{ Ziffern }z\in\{0,1\}}00\ldots$ mit maximal 53 Mantissen-Bits in $S(2, 53)$ exakt darstellbar ist, d.h. $\Delta_{rnd}(T) = 0$.

$X = (7, 8)$
Man erkennt auch in diesem Intervall, dass mit den Bitmustern

$$7 < \widetilde{x} := 111.\overbrace{zzzzz\ldots zzzzz}^{49\text{ Ziffern }z\in\{0,1\}}100\ldots < 8$$

die Summe $1+\widetilde{x}$ gegeben ist durch $1+\widetilde{x} = 1000.\overbrace{zzzzz\ldots zzzzz}^{49\text{ Ziffern }z\in\{0,1\}}100\ldots$, woraus sich der maximale Rundungsfehler $\Delta_{rnd}(T) := 2^{-50}$ unmittelbar ergibt.

Die bisherigen Intervalle $(1, 2)$ bis $(7, 8)$ können verallgemeinert werden:

$$\widetilde{x} \in (\texttt{pred}(2^n), 2^n) \implies \Delta_{rnd}(T) = 2^{n-53}, \quad n = 1, 2, \ldots 52; \tag{3.13}$$

$$\widetilde{x} \in [2^n, \texttt{pred}(2^{n+1})] \implies \Delta_{rnd}(T) = 0, \quad n = 1, 2, \ldots 52; \tag{3.14}$$

Wir beweisen (3.13) für $n = 52$: Mit den möglichen Bitmustern

$$\texttt{pred}(2^{52}) \le \widetilde{x} := \overbrace{11111\ldots11111}^{52\text{ Ziffern }1}.\,z < 2^{52}, \quad z \in \{0, 1\}$$

$$1 + \widetilde{x} = 1\overbrace{00000\ldots00000}^{52\text{ Ziffern }0}.\,z00\ldots$$

gilt für die Summe , und für $z = 1$ ergibt sich der maximale Rundungsfehler $\Delta_{rnd}(T) = 2^{-1}$.

Wir beweisen (3.14) für $n = 52$: Mit den möglichen Bitmustern für $\widetilde{x}$

$$2^{52} \le \widetilde{x} := 1\overbrace{zzzzz\ldots zzzzz}^{52\text{ Ziffern }z\in\{0,1\}}.000\ldots \le \mathtt{pred}(2^{53})$$

zeigt man direkt, dass die Summe $1+\widetilde{x}$ mit maximal 53 Mantissen-Bits in $S(2,53)$ exakt darstellbar ist.

Für alle noch fehlenden Intervalle gilt allgemein:

$$(3.15)\qquad \widetilde{x} \in [2^n, \mathtt{pred}(2^{n+1})] \quad\Longrightarrow\quad \Delta_{rnd}(T) = 2^{n-52} - 1, \quad n = 53, 54\ldots$$

Wir beweisen (3.15) für $n = 53$: Mit den möglichen Bitmustern für $\widetilde{x}$

$$2^{53} \le \widetilde{x} := 1\overbrace{zzzzz\ldots zzzzz}^{52\text{ Ziffern }z\in\{0,1\}}0.000\ldots \le \mathtt{pred}(2^{54})$$

erhält man für die Summe $1+\widetilde{x} = 1\overbrace{zzzzz\ldots zzzzz}^{52\text{ Ziffern }z\in\{0,1\}}1.000$, und daraus ergibt sich direkt die absolute Fehlerschranke $\Delta_{rnd}(T) = 2^{53-52} - 1 = 1$.

Wir beweisen (3.15) noch für $n = 54$: Mit den möglichen Bitmustern für $\widetilde{x}$

$$2^{54} \le \widetilde{x} := 1\overbrace{zzzzz\ldots zzzzz}^{52\text{ Ziffern }z\in\{0,1\}}00.000\ldots \le \mathtt{pred}(2^{55})$$

erhält man für die Summe $1+\widetilde{x} = 1\overbrace{zzzzz\ldots zzzzz}^{52\text{ Ziffern }z\in\{0,1\}}01.000$; der maximale Abstand von $1+\widetilde{x}$ zur nächsten Rasterzahl, d.h. $\Delta_{rnd}(T)$ ist dann gegeben durch:

$$\begin{array}{rcr} & & 1\overbrace{zzzz\ldots zzzz}^{52\text{ Ziffern}z\in\{0,1\}}\ 00.00\ldots \\ & + & 1\ 00.00\ldots \\ & - & \underline{1\ zzzz\ldots zzzz\ 01.00\ldots} \\ \Delta_{rnd}(T) & = & 000\ 11.00\ldots = 2+1 = 3 \end{array}$$

Beachten Sie bitte, dass die Summe aus den beiden ersten Summanden die bez. der exakten Summe $1+\widetilde{x}$ rechts liegende Rasterzahl ist, die zu $1+\widetilde{x}$ den maximalen Abstand $\Delta_{rnd}(T) = 3$ hat. Der obige erste Summand ist die bez. der exakten Summe $1+\widetilde{x}$ links liegende Rasterzahl mit dem kleineren Abstand[4] $01.00 = 1$.

Für $n = 55, 56, 57, 58, \ldots$ lassen sich die in (3.15) angegebenen Fehlerschranken $\Delta_{rnd}(T) = 2^{n-52} - 1$ ganz analog beweisen ∎

Wir kommen jetzt wieder zurück zur Abschätzung des absoluten Rundungsfehlers $\Delta_{rnd}(T)$ bei Auswertung des Terms $T(\widetilde{x}) := 1+\widetilde{x} \approx \widetilde{T}(\widetilde{x}) := 1 \oplus \widetilde{x}$.

[4]Dieser kleinere Abstand 1 wäre bei einer maximal genauen Arithmetik die gesuchte absolute Fehlerschranke.

Zu einem vorgegebenen Maschinenintervall `X` liefert die Funktion `Rnd_1pxh(X)` aus dem **C-XSC** Modul `bnd_arh` für alle $\widetilde{x} \in$ `X` eine garantierte Oberschranke `bnd` des absoluten Rundungsfehlers $\Delta_{rnd}(T)$, d.h. es gilt:

$$\Delta_{rnd}(T) := |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq \texttt{bnd} \quad \text{für alle } \widetilde{x} \in \texttt{X}, \quad T(\widetilde{x}) := 1 + \widetilde{x}; \tag{3.16}$$

Das folgende Programm `book_expl11.cpp`

```
#include "bnd_arh.hpp" // Wegen Rnd_1pxh()
#include <iostream>    // Wegen cout

using namespace cxsc;
using namespace std;
// Berechnung des abs. Rundungsfehlers bei Auswertung von
// T(x) = 1+x in nur hochgenauer Arithmetik für x aus [0,1].

int main()
{
    interval X(0,1);    // X = [0,1]
    real bnd;
    bnd = Rnd_1pxh(X);  // Rundungsfehler bzgl. T(x) = 1+x;
    cout << RndUp << "Abs. Rundungsfehler = " << bnd << endl;
}
```

liefert für `X` $= [0, 1]$ nach Tabelle 3.1 die Bildschirmausgabe:

```
Abs. Rundungsfehler = 2.220447E-016
```

Nach (3.7) sind wir jetzt in der Lage, den absoluten Gesamtfehler abzuschätzen, wenn der Term $T(x) = 1 + x$ für $x \in$ `X` in nur hochgenauer Arithmetik ausgewertet wird. Die reellen $x \in$ `X` interpretieren wir wieder als Funktionswerte eines Wertebereichs `X`. Diese Funktionswerte seien auf der Maschine fehlerhaft berechnet, und die Näherungswerte $\widetilde{x}$ sollen eine der auf Seite 57 angegebenen Ungleichungen (3.5) oder (3.6) erfüllen. Für den Term $T(x) = 1+x$ kann dann der absolute Gesamtfehler $|T(x) - \widetilde{T}(\widetilde{x})|$ mit einer der beiden Funktionen aus dem **C-XSC** Modul `bnd_arh` abgeschätzt werden:

`abs_1px_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`abs_1px_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Ganz entsprechend kann der relative Gesamtfehler $|(T(x) - \widetilde{T}(\widetilde{x}))/T(x)|$ mit Hilfe der beiden folgenden Funktionen abgeschätzt werden:

`rel_1px_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`rel_1px_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Die `real`-Variable `bnd` liefert jeweils eine garantierte Oberschranke für den absoluten oder relativen Gesamtfehler, und `T` ist eine Einschließung von 1+`X`, d.h. für alle $x \in$ `X` liegen die Summen $1 + x$ stets im Maschinenintervall `T`.

Beachten Sie bitte, dass bei der Auswertung von $T(x)$ auf der Maschine nur die Summen $\widetilde{T}(\widetilde{x}) = 1 \oplus \widetilde{x} \in S(2,53)$ der beiden Maschinenzahlen 1 und $\widetilde{x}$ gebildet werden und dass die Funktion `abs_1px_delh(X,del,T,bnd)` mit `bnd` für alle $x \in$ `X` eine Oberschranke des absoluten Gesamtfehlers $|T(x) - \widetilde{T}(\widetilde{x})|$ liefert.

3.2.2 $T(x) = 1 - x$

Wegen $1 - x \equiv 1 + (-x)$ können für den Term $T(x) = 1 - x$ die Schranken der absoluten und relativen Gesamtfehler für alle $x \in$ `X` mit Hilfe der in Abschnitt 3.2.1 angegebenen Funktionen berechnet werden, wenn man dort das Eingangsintervall `X` durch $-$`X` ersetzt. Der Anwender muss diese Substitution jedoch nicht selbst ausführen; ihm stehen vielmehr eigens für den Term $T(x) = 1 - x$ neu benannte **C-XSC** Funktionen zu Verfügung. Wir interpretieren wieder die reellen $x \in$ `X` als die Funktionswerte eines Wertebereichs `X`. Diese exakten Funktionswerte $x \in$ `X` seien auf der Maschine als Rasterzahlen $\widetilde{x} \in S(2,53)$ fehlerhaft berechnet worden, wobei eine der beiden Ungleichungen (3.5) oder (3.6) auf Seite 57 erfüllt sein soll.

Bei nur hochgenauer Arithmetik wird der absolute Gesamtfehler $|T(x) - \widetilde{T}(\widetilde{x})|$ abgeschätzt mit Hilfe der im Modul `bnd_arh` implementierten Funktionen:

`abs_1mx_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`abs_1mx_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Bei nur hochgenauer Arithmetik wird der relative Gesamtfehler $|(T(x) - \widetilde{T}(\widetilde{x}))/T(x)|$ abgeschätzt mit Hilfe der im Modul `bnd_arh` implementierten Funktionen:

`rel_1mx_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`rel_1mx_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Die `real`-Variable `bnd` liefert jeweils eine garantierte Oberschranke für den absoluten oder relativen Gesamtfehler, und `T` ist eine Einschließung von 1$-$`X`, d.h. für alle $x \in$ `X` liegen die Differenzen $1 - x$ stets im Maschinenintervall `T`.

Beachten Sie bitte, dass bei der Auswertung von $T(x)$ auf der Maschine nur die Differenzen $\widetilde{T}(\widetilde{x}) = 1 \ominus \widetilde{x} \in S(2,53)$ der beiden Maschinenzahlen 1 und $\widetilde{x}$ gebildet werden und dass die Funktion `abs_1mx_delm(X,del,T,bnd)` mit `bnd` für alle $x \in$ `X` bei maximal genauer Arithmetik eine Oberschranke des absoluten Gesamtfehlers $|T(x) - \widetilde{T}(\widetilde{x})|$ liefert. Für alle $x \in$ `X` wird dabei vorausgesetzt: $|x - \widetilde{x}| \leq \Delta(x) \leq$ `del`.

3.2.3 $T(x) = x - 1$

Wegen $x - 1 \equiv -(1 + (-x))$ können für den Term $T(x) = x - 1$ die Schranken der absoluten und relativen Gesamtfehler für alle $x \in$ `X` mit Hilfe der in Abschnitt 3.2.1 angegebenen Funktionen berechnet werden, wenn man dort das Eingangsintervall `X` durch $-$`X` ersetzt und das Ausgangsintervall `T` mit -1 multipliziert. Der Anwender muss diese Substitutionen jedoch nicht selbst ausführen; ihm stehen vielmehr eigens für den Term $T(x) = x - 1$ neu benannte **C-XSC** Funktionen zu Verfügung. Wir interpretieren wieder die reellen $x \in$ `X` als die Funktionswerte eines Wertebereichs `X`. Diese exakten Funktionswerte $x \in$ `X` seien auf der Maschine als Rasterzahlen $\widetilde{x} \in S(2, 53)$ fehlerhaft berechnet worden, wobei eine der beiden Ungleichungen (3.5) oder (3.6) auf Seite 57 erfüllt sein soll.

Bei nur hochgenauer Arithmetik wird der absolute Gesamtfehler $|T(x) - \widetilde{T}(\widetilde{x})|$ abgeschätzt mit Hilfe der im Modul `bnd_arh` implementierten Funktionen:

`abs_xm1_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`abs_xm1_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Bei nur hochgenauer Arithmetik wird der relative Gesamtfehler $|(T(x) - \widetilde{T}(\widetilde{x}))/T(x)|$ abgeschätzt mit Hilfe der im Modul `bnd_arh` implementierten Funktionen:

`rel_xm1_delh(X,del,T,bnd)`,	falls $\Delta(x) \leq$ `del` gegeben ist.
`rel_xm1_epsh(X,eps,T,bnd)`,	falls $\varepsilon(x) \leq$ `eps` gegeben ist.

Die `real`-Variable `bnd` liefert jeweils eine garantierte Oberschranke für den absoluten oder relativen Gesamtfehler, und `T` ist eine Einschließung von `X`-1, d.h. für alle $x \in$ `X` liegen die Differenzen $x - 1$ stets im Maschinenintervall `T`.

Beachten Sie bitte, dass bei der Auswertung von $T(x)$ auf der Maschine nur die Differenzen $\widetilde{T}(\widetilde{x}) = \widetilde{x} \ominus 1 \in S(2, 53)$ der beiden Maschinenzahlen $\widetilde{x}$ und 1 gebildet werden und dass die Funktion `abs_xm1_delm(X,del,T,bnd)` mit `bnd` für alle $x \in$ `X` bei maximal genauer Arithmetik eine Oberschranke des absoluten Gesamtfehlers $|T(x) - \widetilde{T}(\widetilde{x})|$ liefert. Für alle $x \in$ `X` wird dabei vorausgesetzt: $|x - \widetilde{x}| \leq \Delta(x) \leq$ `del`.

3.2.4 Implementierung einer Intervallfunktion

Wie in Abschnitt 2.5.4 wollen wir wieder für den Term

$$T(x) := \frac{1+x}{1-x}$$

zunächst den absoluten Fehler unter folgenden Voraussetzungen berechnen:

1. Die in $T(x)$ auftretenden Grundoperationen $+, -, /$ sollen auf der Maschine in nur hochgenauer Arithmetik ausgeführt werden.
2. Für die exakten Werte $x \in I\!R$ soll gelten: $x \in \mathtt{X} := [-10, +0.5]$.
3. Da nicht alle $x \in \mathtt{X}$ als Maschinenzahlen $\widetilde{x} \in S(2, 53)$ darstellbar sind, wird vorausgesetzt, dass ein solches $x \in \mathtt{X}$ stets zur nächsten Rasterzahl $\widetilde{x}$ gerundet wird. Der durch diese Rundung bedingte absolute Fehler $|x - \widetilde{x}|$ lässt sich dann abschätzen durch $|x - \widetilde{x}| \leq |\mathtt{X}| \cdot \varepsilon(m) = 10 \cdot \varepsilon(m) = 10 \cdot 2^{-53} = \Delta(x)$. Für alle $x \in \mathtt{X}$ soll damit gelten:

$$|x - \widetilde{x}| \leq \Delta(x) = 10 \cdot 2^{-53} = 1.11022302\ldots \cdot 10^{-15}, \quad x \in \mathtt{X} = [-10, +0.5]$$

In diesem Abschnitt sollen jetzt jedoch zur Verbesserung der Fehlerschranken von Zähler und Nenner die in den Abschnitten 3.2.1 und 3.2.2 angegebenen Funktionen benutzt werden. Bezeichnen wir mit $\widetilde{T}(\widetilde{x}) := (\widetilde{x} \ominus 1) \oslash (\widetilde{x} \oplus 1)$ den auf der Maschine ausgewerteten Term $T(x)$, so berechnet das nachfolgende Programm eine Oberschranke `del` des dabei aufgetretenen absoluten Fehlers $|T(x) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del}$.

```
#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_arh.hpp"  // Wegen abs_1px_delh(),abs_1mx_delh()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
int main()
{
    interval X(-10,0.5),Z,N,T;
    real delx,delz,deln,del;
    cin >> RndUp; string("1.11022303E-15") >> delx;
    abs_1px_delh(X,delx,Z,delz);  // delz: abs. Fehler für 1+x
    abs_1mx_delh(X,delx,N,deln);  // deln: abs. Fehler für 1-x
    abs_divh1(Z,delz,N,deln,T,del);
             // del : absoluter Gesamtfehler bzgl. (1+x)/(1-x)
    cout << RndUp << "Abs. Fehlerschranke = " << del << endl;
    cout << "(1+x)/(1-x) enthalten in:  " << T << endl;
}
```

Das umseitige Programm `book_expl12.cpp` liefert die folgende Bildschirmausgabe:

```
Abs. Fehlerschranke = 7.860380E-014
(1+x)/(1-x) enthalten in: [-18.000000,  3.000000]
```

Anmerkungen:

1. Für alle $x \in \mathtt{X} = [-10, +0.5]$ gilt damit für die Abschätzung des absoluten Gesamtfehlers $|T(x) - \widetilde{T}(\widetilde{x})|$:

   $$|T(x) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del} \leq 7.860380 \cdot 10^{-14}$$

   Vergleicht man mit dem entsprechenden Wert $1.381118 \cdot 10^{-13}$ von Seite 43, so erkennt man eine deutliche Verbesserung des absoluten Gesamtfehlers.

2. Wählt man $\mathtt{delx} = 0$, so liefert das Programm die natürlich kleinere absolute Fehlerschranke

   $$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq \mathtt{del} \leq 3.641532 \cdot 10^{-14} \quad \text{für alle } \widetilde{x} \in \mathtt{X} = [-10, +0.5]$$

   Beachten Sie bitte, dass jetzt das Intervall $\mathtt{X} = [-10, +0.5]$ wegen $\mathtt{delx} = 0$ nur die exakten Maschinenzahlen $\widetilde{x} \in \mathtt{X}$ enthält und dass sich obige Fehlerschranke nur auf diese Maschinenzahlen bezieht!

3. Wenn man jetzt unter der gleichen Voraussetzung $\mathtt{delx} = 0$ das Intervall $\mathtt{X}$ in die beiden Teilintervalle $\mathtt{X_1} = [-10, -5]$ und $\mathtt{X_2} = [-5, +0.5]$ unterteilt und für beide Teilintervalle den jeweiligen absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ extra berechnet, so erhält man die Ergebnisse:

   $$\begin{aligned} |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| &\leq \mathtt{del1} \leq 5.551116 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in \mathtt{X_1} = [-10, -5] \\ |T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| &\leq \mathtt{del2} \leq 9.325874 \cdot 10^{-15} \quad \text{für alle } \widetilde{x} \in \mathtt{X_2} = [-5, +0.5] \end{aligned}$$

Nach Punkt 2 der letzten Anmerkungen gilt: $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq 3.641532 \cdot 10^{-14}$ für alle $\widetilde{x} \in \mathtt{X} = [-10, +0.5]$. Vergleicht man mit der absoluten Fehlerschranke aus Punkt 3, so erhält man eine etwa um den Faktor 4 kleinere Fehlerschranke, wenn man das Intervall $\mathtt{X} = [-10, +0.5]$ in nur zwei Teilintervalle $\mathtt{X_1}$ und $\mathtt{X_2}$ unterteilt.

Im nächsten Programm wollen wir für den gleichen Term $T(x) = (1+x)/(1-x)$ über dem gleichen Intervall $\mathtt{X} = [-10, +0.5]$ unter der Voraussetzung $\Delta(x) = 0$ den absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ möglichst optimal abschätzen, indem wir das Intervall `X` in hinreichend viele Teilintervalle `Xi` zerlegen und für jedes Teilintervall `Xi` den absoluten Fehler einzeln abschätzen. Der absolute Fehler über dem Gesamtintervall `X` ist dann das Maximum der absoluten Fehler über allen Teilintervallen `Xi`.

```
#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_arh.hpp"  // Wegen abs_1px_delh(),abs_1mx_delh()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Xi, const real& delx)
{
    interval Z,N,T;
    real delz,deln,del;
    abs_1px_delh(Xi,delx,Z,delz); // absl. Fehler bzgl. 1+x
    abs_1mx_delh(Xi,delx,N,deln); // absl. Fehler bzgl. 1-x
    abs_divh1(Z,delz,N,deln,T,del);  // absl. Fehler bzgl. (1+x)/(1-x)
    return del;  // Rückgabe einer Schranke del des absoluten Fehlers.
}

int main()
{
    interval X;
    real bnd, diam = 1e-5, delx=0;
    cout << "Intervall X = ? "; cin >> X;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Absolute Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl13.cpp` liefert bei Eingabe von `X` $= [-10, 0.5]$ den Bildschirmausdruck:

```
Absolute Fehlerschranke = 1.443290E-015
```

Anmerkungen:

1. Wegen $\Delta(x) =$ `delx` $= 0$ bezieht sich die letzte Fehlerschranke nur auf die Maschinenzahlen $\widetilde{x} \in$ `X`. Für den absoluten Fehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ gilt daher bei nur hochgenauer Arithmetik die Abschätzung:

   $$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq 1.443290 \cdot 10^{-15} \quad \text{für alle } \widetilde{x} \in \texttt{X} = [-10, +0.5]$$

   Vergleicht man mit den Fehlerschranken aus Punkt 2 von Seite 73, so erkennt man, das der absolute Fehler durch die sehr feine Intervallunterteilung etwa um den Faktor 25 verkleinert werden konnte.

2. Beachten Sie bitte, dass in der Funktion `main()` der Manipulator `RndUp` in der letzten Zeile nicht vergessen werden darf, da durch ihn die binäre Fehlerschranke `bnd` zur nächstgrößeren Dezimalzahl des gewählten Ausgabeformats aufgerundet wird. Bei fehlendem Manipulator `RndUp` erfolgt die Rundung zur nächsten Dezimalzahl, was eventuell mit einer Abrundung verbunden ist. In einem solchen Fall kann der ausgegebene Dezimalwert dann nicht mehr als garantierte Oberschranke des absoluten Fehlers angesehen werden!

3. Die Funktion `T_x()` liefert für $T(x) := (1+x)/(1-x)$ eine Oberschranke des absoluten Fehlers $|T(x) - \widetilde{T}(\widetilde{x})|$ für alle $x \in$ `X`, wenn für die gestörten Maschinenzahlen $\widetilde{x}$ gilt: $|x - \widetilde{x}| \leq \Delta(x) =$ `delx`.

4. Die Funktion `Max_bnd_Xi()` aus dem **C-XSC** Modul `bnd_util` zerlegt das Intervall `X` in Teilintervalle `Xi`, deren Durchmesser durch den übergebenen Wert `diam` bestimmt ist, wobei `diam` $= 10^{-5}$ meist eine gute Wahl ist. Mit Hilfe von `T_x()` wird für jedes Teilintervall `Xi` die durch `T_x()` bestimmte Fehlerschranke abgeschätzt und das Maximum all dieser Fehlerschranken in der `real`-Variablen `bnd` gespeichert. Der hier an `Max_bnd_Xi()` übergebene Parameter `delx` wird intern weiter an `T_x()` übergeben und bestimmt die absolute Fehlerschranke der gestörten Werte $\widetilde{x}$, d.h. es gilt: $|x - \widetilde{x}| \leq \Delta(x) =$ `delx` für alle $x \in$ `X`.

Wir wollen jetzt noch für den gleichen Term $T(x) := (1+x)/(1-x)$ eine Oberschranke des relativen Fehlers $|(T(x)-\widetilde{T}(\widetilde{x}))/T(x)|$ für alle $x \in$ `X` $= [-10, +0.5]$ berechnen. Vorausgesetzt wird wieder $\Delta(x) = 0$, so dass der relative Fehler $(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})$ jetzt also nur noch für alle Maschinenzahlen $\widetilde{x} \in$ `X` $= [-10, +0.5]$ abzuschätzen ist. Wegen $T(-1) = \widetilde{T}(-1) = 0$ ist der relative Fehler für $\widetilde{x} = -1$ unbestimmt und kann deshalb auf Null gesetzt werden. Die Fehlerabschätzung muss daher nur noch in den beiden Intervallen `X1` $= [-10, $`pred`$(-1)]$ und `X2` $=$ [`succ`$(-1), +0.5]$ vorgenommen werden.

Für das Intervall `X1` muss das auf Seite 74 angegebene Programm im Wesentlichen nur in der vorletzten Zeile der Funktion `T_x()` zur Abschätzung des relativen Fehlers bei der Division abgeändert werden:

```
// Programm book_expl14.cpp
#include "abs_relh.hpp" // Wegen rel_divh1()
#include "bnd_arh.hpp"  // Wegen abs_1px_delh(),abs_1mx_delh()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Xi, const real& delx)
```

```
{
    interval Z,N,T;
    real delz,deln,eps;
    abs_1px_delh(Xi,delx,Z,delz); // absl. Fehler bzgl. 1+x
    abs_1mx_delh(Xi,delx,N,deln); // absl. Fehler bzgl. 1-x
    rel_divh1(Z,delz,N,deln,T,eps);  // rel. Fehler bzgl. (1+x)/(1-x)
    return eps;  // Rückgabe einer Schranke eps des relativen Fehlers.
}

int main()
{
    interval X1 = interval(-10,pred(-1));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X1,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl14.cpp` liefert die folgende Bildschirmausgabe:

```
Relative Fehlerschranke = 3.330670E-016
```

Anmerkungen:

1. Wegen $\Delta(x) = \texttt{delx} = 0$ bezieht sich die obige Fehlerschranke nur auf die Maschinenzahlen $\widetilde{x} \in \texttt{X1} = [-10, \texttt{pred}(-1)]$. Für den gesuchten relativen Fehler $|(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})|$ gilt daher bei vorausgesetzter hochgenauer Arithmetik die Abschätzung:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq 3.330670 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in \texttt{X1} = [-10, \texttt{pred}(-1)]$$

2. Ändert man im obigen Programm `book_expl14.cpp` in der Funktion `main()` die drei ersten Zeilen wie folgt

```
interval X2 = interval(succ(-1),+0.5);
real bnd, diam = 1e-5, delx=0;
Max_bnd_Xi(T_x,X2,delx,diam,bnd);
```

   so erhält man für den relativen Fehler $|(T(\widetilde{x}) - \widetilde{T}(\widetilde{x}))/T(\widetilde{x})|$ die Abschätzung:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq 5.551116 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in \texttt{X2} = [\texttt{succ}(-1), +0.5]$$

3. Im gesamten Intervall $\mathtt{X} = [-10, +0.5]$ gilt dann für den relativen Fehler die Abschätzung:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq \varepsilon(T) = 5.551116 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in \mathtt{X} = [-10, +0.5]$$

Wir hatten dabei $\Delta(x) = \mathtt{delx} = 0$ vorausgesetzt, so dass für alle $x \in \mathtt{X}$ gilt: $x = \widetilde{x}$. Die obige Fehlerabschätzung bezieht sich daher nur auf die Rasterzahlen $\widetilde{x} \in \mathtt{X} = [-10, +0.5]$, d.h. $\widetilde{x} \in \mathtt{X} \cap S(2, 53)$.

4. Vergleicht man die gefundene relative Fehlerschranke $5.551116 \cdot 10^{-16}$ mit dem auf Seite 47 gefundenen Wert $2.000001 \cdot 10^{-5}$, so erkennt man, dass mit Hilfe der verbesserten Schranken für die Terme $1 + x$ und $1 - x$ die relative Fehlerschranke um ganze 11 Zehnerpotenzen reduziert werden konnte! An Stelle der allgemeinen Funktionen aus dem Modul `abs_relh` zur Abschätzung der absoluten oder relativen Fehler bei Addition und Subtraktion sollte man daher bei den Termen $1 + x,\ 1 - x$ und $x - 1$ stets auf die entsprechenden Funktionen aus dem Modul `bnd_arh` zurückgreifen!

5. Benutzt man bei den drei Termen $1 + x,\ 1 - x$ und $x - 1$ die entsprechenden Funktionen aus dem Modul `bnd_arh`, so ist eine zusätzliche Anwendung der Sätze 3.1.1 und 3.1.2 von Seite 50 nicht nötig, da die genannten Funktionen schon optimale Fehlerschranken liefern.

Für den Term

$$T(x) = \frac{1 + x}{1 - x}, \quad x \in \mathtt{X} \in IR$$

hatten wir unter der Voraussetzung $|x - \widetilde{x}| \leq \Delta(x) = 0$ nach Anmerkung 1 auf Seite 74 eine Schranke für den absoluten Gesamtfehler berechnet. Wegen $\Delta(x) = 0$ kann das gegebene Maschinenintervall $\mathtt{X}$ nur die endlich vielen Rasterzahlen $\widetilde{x} \in S(2, 53)$ enthalten, so dass sich die angegebene Fehlerschranke nur auf diese Maschinenzahlen $\widetilde{x} \in \mathtt{X} = [-10, 0.5]$ beziehen kann. Für den absoluten Gesamtfehler $|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})|$ hatten wir daher gefunden:

$$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq \Delta(T) = 1.443290 \cdot 10^{-15} \quad \text{für alle } \widetilde{x} \in \mathtt{X} \cap S(2, 53).$$

Für alle $\widetilde{x} \in \mathtt{X} \cap S(2, 53)$ gilt damit:

$$\widetilde{T}(\widetilde{x}) - \Delta(T) \leq T(\widetilde{x}) \leq \widetilde{T}(\widetilde{x}) + \Delta(T) \tag{3.17}$$

Obige Einschließung für $T(\widetilde{x})$ gilt zunächst leider nur für alle Maschinenzahlen $\widetilde{x} \in \mathtt{X}$. Es wäre jedoch wünschenswert, wenn man eine Einschließung der Termwerte $T(x)$ für alle reellen $x \in \mathtt{X}_0 \in IR$ berechnen könnte, wenn $\mathtt{X}_0 = [\widetilde{x}_1, \widetilde{x}_2] \subseteq \mathtt{X}$ ein beliebig vorgegebenes Maschinenintervall ist.

Eine solche Einschließung lässt sich mit Hilfe von $\Delta(T)$ sehr einfach angeben, wenn man beachtet, dass $T(x)$ wegen $T'(x) > 0$ in ganz $\mathtt{X}$ streng monoton wächst. Für alle reellen $x \in [\widetilde{x}_1, \widetilde{x}_2]$ gilt daher: $T(\widetilde{x}_1) \leq T(x) \leq T(\widetilde{x}_2)$, und mit (3.17) erhält man direkt:

$$\widetilde{T}(\widetilde{x}_1) - \Delta(T) \leq T(x) \leq \widetilde{T}(\widetilde{x}_2) + \Delta(T), \quad x \in [\widetilde{x}_1, \widetilde{x}_2] \tag{3.18}$$

Zur Einschließung aller exakten Termwerte $T(x)$ benötigt man also lediglich die auf der Maschine fehlerhaft berechneten Werte $\widetilde{T}(\widetilde{x}_1), \widetilde{T}(\widetilde{x}_2)$ und die absolute Fehlerschranke $\Delta(T)$. Beachten Sie bitte, dass wir wegen der Monotonie von $T(x)$ jetzt mit (3.18) nicht nur Einschließungen der endlich vielen $T(\widetilde{x})$ berechnen, sondern dass wir mit (3.18) den ganzen Bereich $[T(\widetilde{x}_1), T(\widetilde{x}_2)]$ einschließen! Mit Hilfe der absoluten Fehlerschranke $\Delta(T)$ kann man daher bei monotonen Funktionen die entsprechenden Intervallfunktionen sehr einfach implementieren.

Insbesondere bei sehr breiten Intervallen $\mathtt{X} \in IR$ erhält man für $[T(\widetilde{x}_1), T(\widetilde{x}_2)]$, mit $x \in [\widetilde{x}_1, \widetilde{x}_2]$, i.a. bessere Einschließungen, wenn man die auf Seite 77 angegebene relative Fehlerschranke $\varepsilon(T) = 5.551116 \cdot 10^{-16}$ benutzt. Mit

$$I(\widetilde{x}) := \begin{cases} \widetilde{T}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right) & \text{falls } \widetilde{T}(\widetilde{x}) \geq 0 \\ \widetilde{T}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right) & \text{falls } \widetilde{T}(\widetilde{x}) < 0 \end{cases} \tag{3.19}$$

$$S(\widetilde{x}) := \begin{cases} \widetilde{T}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right) & \text{falls } \widetilde{T}(\widetilde{x}) \geq 0 \\ \widetilde{T}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right) & \text{falls } \widetilde{T}(\widetilde{x}) < 0 \end{cases} \tag{3.20}$$

gilt dann für die monoton wachsende Funktion $T(x)$

$$I(\widetilde{x}_1) \leq T(x) \leq S(\widetilde{x}_2), \quad \text{für alle } x \in [\widetilde{x}_1, \widetilde{x}_2] \subseteq \mathtt{X} \tag{3.21}$$

Für eine monoton fallende Funktion $T(x)$ gilt ganz entsprechend

$$I(\widetilde{x}_2) \leq T(x) \leq S(\widetilde{x}_1), \quad \text{für alle } x \in [\widetilde{x}_1, \widetilde{x}_2] \subseteq \mathtt{X} \tag{3.22}$$

Wir beweisen in (3.21) zunächst die zweite Ungleichung für eine monoton wachsende Funktion $T(x)$. Dann gilt zunächst $T(x) \leq T(\widetilde{x}_2)$, und für $T(\widetilde{x}_2)$ benötigen wir jetzt eine garantierte Oberschranke. Ausgangspunkt dazu ist die Definition des relativen Fehlers ε_T:

$$\varepsilon_T := \frac{\widetilde{T}(\widetilde{x}) - T(\widetilde{x})}{T(\widetilde{x})} \quad \text{bzw.} \quad T(\widetilde{x}) = \widetilde{T}(\widetilde{x}) \cdot \frac{1}{1+\varepsilon_T} \tag{3.23}$$

Vorausgesetzt wird dabei

$$|\varepsilon_T| \leq \varepsilon(T) \; \forall \widetilde{x} \in [\widetilde{x}_1, \widetilde{x}_2], \quad \text{d.h. es gilt:} \quad \varepsilon_T \in [-\varepsilon(T), +\varepsilon(T)]. \tag{3.24}$$

Für $T(\widetilde{x}_2) \geq 0$, d.h. $\widetilde{T}(\widetilde{x}_2) \geq 0$ erhält man dann direkt:

$$T(x) \leq T(\widetilde{x}_2) \leq \widetilde{T}(\widetilde{x}_2) \cdot \frac{1}{1-\varepsilon(T)} = \widetilde{T}(\widetilde{x}_2) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right)$$

und für $T(\widetilde{x}_2) < 0$, d.h. $\widetilde{T}(\widetilde{x}_2) < 0$ erhält man ganz entsprechend:

$$T(x) \leq T(\widetilde{x}_2) \leq \widetilde{T}(\widetilde{x}_2) \cdot \frac{1}{1+\varepsilon(T)} = \widetilde{T}(\widetilde{x}_2) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right)$$

und damit haben wir $T(x) \leq S(\widetilde{x}_2)$ schon bewiesen.

Zum Beweis der ersten Ungleichung in (3.21) gilt wegen der Monotonie zunächst wieder $T(\widetilde{x}_1) \leq T(x)$, und eine Unterschranke zu $T(\widetilde{x}_1)$ ergibt sich aus (3.23) zu:

$$\widetilde{T}(\widetilde{x}_1) \cdot \frac{1}{1+\varepsilon(T)} = \widetilde{T}(\widetilde{x}_1) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right) \leq T(\widetilde{x}_1) \leq T(x), \text{ falls } T(\widetilde{x}_1) \geq 0.$$

Im Falle $\widetilde{T}(\widetilde{x}_1) < 0$ und damit auch $T(\widetilde{x}_1) < 0$ erhält man ganz analog:

$$\widetilde{T}(\widetilde{x}_1) \cdot \frac{1}{1-\varepsilon(T)} = \widetilde{T}(\widetilde{x}_1) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right) \leq T(\widetilde{x}_1) \leq T(x), \text{ falls } T(\widetilde{x}_1) < 0$$

und damit haben wir auch $I(\widetilde{x}_1) \leq T(x)$ bewiesen, es gilt damit für die monoton wachsende Funktion $T(x)$ die Einschließung:

$$I(\widetilde{x}_1) \leq T(x) \leq S(\widetilde{x}_2), \quad \text{für alle } x \in [\widetilde{x}_1, \widetilde{x}_2] \subseteq \mathtt{X}$$

Die beiden Ungleichungen in (3.22) beweist man für eine monoton fallende Funktion ganz analog.

Zusammenfassung:

Hat man für einen Term $T(x)$, mit $x \in \mathtt{X} \in IR$, unter der Voraussetzung $\Delta(x) = 0$ bzw. $\varepsilon(x) = 0$ eine absolute oder relative Fehlerschranke $\Delta(T)$ oder $\varepsilon(T)$ berechnet, so beziehen sich diese Fehlerschranken zunächst nur auf alle Rasterzahlen $\widetilde{x} \in \mathtt{X}$. Ist $T(x)$ zusätzlich monoton in $\mathtt{X}$, so können mit Hilfe von $I(\widetilde{x})$ und $S(\widetilde{x})$ nach (3.21) bzw. (3.22) Einschließungen für $T(x)$ **für alle** $x \in \mathtt{X}$ angegeben werden.

Im folgenden Programm `book_expl15.cpp` findet man für $T(x) = (1+x)/(1-x)$, mit $x \in$ `X` $= [-10, +0.5]$ und $\varepsilon(T) = 5.551116 \cdot 10^{-16}$ die Implementierung der Intervallfunktion `interval T_x(const interval& x)`.

```
// Program  book_expl15.cpp
#include "bnd_util.hpp" // Wegen eps2fractions()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const real& x)
{  // point function for T(x)
    return (1+x)/(1-x);
}
interval T_x(const interval& x)
{   // Intervallfunktion des monoton wachsenden Terms T(x)
    string str = "5.551116E-16"; // Rel. Fehlerschranke von T(x)
    real Z1p,N1p,Z1m,N1m;
    eps2fractions(str,Z1p,N1p,Z1m,N1m);
    real x1 = T_x(Inf(x)), x2 = T_x(Sup(x));
    x1 *= (sign(x1)<0) ? Z1p/N1p : Z1m/N1m;
    x2 *= (sign(x2)<0) ? Z1m/N1m : Z1p/N1p;
    return interval(x1,x2); // Einschließung von T(x)
}
int main()
{
    interval y,x;
    cout << "Intervall aus [-10,+0.5] = ? "; cin >> x;
    y = T_x(x);
    cout << SetPrecision(15,15)
         << "Einschlg. von T(x) =  " << y << endl;
}
```

Mit der Intervalleingabe $x = [-2, +0.5]$ erhält man die folgende Bildschirmausgabe:

```
Einschlg. von T(x) =  [-0.333333333333334,3.000000000000004]
```

Anmerkungen:

1. Das Intervall `[-0.333333333333334,3.000000000000004]` enthält damit bewiesenermaßen alle Termwerte $T(x)$, mit $x \in [-2, 0.5]$.

2. Wegen der in **C++**möglichen Funktionsüberladung haben die beiden Funktionen mit dem `real`- bzw. `interval`-Argument `x` den gleichen Namen `T_x`, womit die Lesbarkeit von Programmen deutlich erleichtert wird.

3. Der Funktionsaufruf `eps2fractions(str,Z1p,N1p,Z1m,N1m);` liefert zum String `str = "5.551116E-16"` Zähler und Nenner der Quotienten `Z1p/N1p` und `Z1m/N1m`, wobei im Falle $\widetilde{T}(\widetilde{x}) \geq 0$ die Ungleichungen

$$\begin{aligned} \widetilde{T}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right) &\leq \widetilde{T}(\widetilde{x}) \odot (\texttt{Z1p} \oslash \texttt{N1p}) \\ \widetilde{T}(\widetilde{x}) \odot (\texttt{Z1m} \oslash \texttt{N1m}) &\leq \widetilde{T}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right) \end{aligned}$$

   erfüllt sind. Im Falle $\widetilde{T}(\widetilde{x}) < 0$ gilt ganz entsprechend:

$$\begin{aligned} \widetilde{T}(\widetilde{x}) \odot (\texttt{Z1p} \oslash \texttt{N1p}) &\leq \widetilde{T}(\widetilde{x}) \cdot \left(1 + \frac{\varepsilon(T)}{1-\varepsilon(T)}\right) \\ \widetilde{T}(\widetilde{x}) \cdot \left(1 - \frac{\varepsilon(T)}{1+\varepsilon(T)}\right) &\leq \widetilde{T}(\widetilde{x}) \odot (\texttt{Z1m} \oslash \texttt{N1m}) \end{aligned}$$

   Mit Hilfe der Maschinenquotienten `Z1p` $\oslash$ `N1p` und `Z1m` $\oslash$ `N1m` können daher die gesuchten Ober- und Unterschranken des Einschließungsintervalls durch je eine Maschinenmultiplikation, d.h. also ohne die zeitaufwendigen gerichteten Rundungen, berechnet werden!

4. Die zeitaufwendige Berechnung der Quotienten `Z1p` $\oslash$ `N1p` und `Z1m` $\oslash$ `N1m` wird man wie im umseitigen Programmbeispiel natürlich nicht in der Intervallfunktion `T_x()` selbst vornehmen, sondern in entsprechenden **C-XSC** Modulen vorher bereitstellen.

Verbesserung der Fehlerschranke durch Optimierung des Algorithmus:

Für den Term

$$T(x) := \frac{1+x}{1-x}, \qquad x \in [-10, +0.5]$$

hatten wir auf Seite 77 die folgende Schranke für den relativen Fehler gefunden:

$$\left|\frac{\widetilde{T}(\widetilde{x}) - T(\widetilde{x})}{T(\widetilde{x})}\right| \leq \varepsilon(T) = 5.551116 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in \boldsymbol{X} = [-10, +0.5]$$

Wir zeigen jetzt, wie man mit etwas größerem Aufwand diese Fehlerschranke noch deutlich verbessern kann. Da die Werkzeuge zur Fehlerabschätzung im Wesentlichen

ausgereizt sind, bleibt also nur noch eine Optimierung des Algorithmus zur Auswertung von $T(x)$. Wir betrachten dazu ein Teilintervall $[a,b] \subset \boldsymbol{X}$ mit dem Mittelpunkt $x_0 = (a+b)/2$. Die Transformation $x = x_0 + h$ liefert dann[5]:

$$T(x) := \frac{1+x}{1-x} = \frac{(1+x_0)+h}{(1-x_0)-h} = \frac{1+x_0}{1-x_0} + \alpha, \quad \text{mit} \tag{3.25}$$

$$\alpha := \frac{2h}{(1-x_0)\cdot\{(1-x_0)-h\}}, \quad h \in [-\frac{b-a}{2}, \frac{b-a}{2}] \tag{3.26}$$

Wählt man $a, b \in S(2,53)$, so gilt $x_0 \in S(2,53)$, und in speziellen Fällen sind auch $1 - x_0$ und $h := \widetilde{x} - x_0$ exakt darstellbar. Gilt jedoch $(1+x_0)/(1-x_0) \notin S(2,53)$, so berechnet man $(1+x_0)/(1-x_0)$ in doppelter Genauigkeit:

$$\beta := \frac{1+x_0}{1-x_0} = b_1 + b_2 + \delta, \quad \text{mit } b1, b2 \in S(2,53),$$

und muss noch den absoluten Fehler δ abschätzen: $|\delta| \leq \Delta(\beta)$. Wählt man dann den Algorithmus

$$\widetilde{T}(\widetilde{x}) := b_1 \oplus (\widetilde{\alpha} \oplus b_2), \qquad b_1, b_2, \widetilde{\alpha} \in S(2,53)$$

und berücksichtigt die Schranke $\Delta(\beta)$ bei der automatischen Fehlerabschätzung von $(\widetilde{\alpha} \oplus b_2)$, so kann der erste Summand $b_1 \in S(2,53)$ als exakt aufgefasst werden, und bei hinreichend schmalen Intervallen $[a,b]$ erhält man deutlich kleinere $\varepsilon(T)$-Werte.

Als Beispiel betrachten wir den besonders einfachen Fall $[a,b] = [-0.2, +0.2]$ und $x_0 = 0$. Wegen $x \equiv h$ gilt dann:

$$T(x) = 1 + \frac{2x}{1-x},$$

wobei der Zähler $2x$ für alle Maschinenzahlen $x = \widetilde{x}$ exakt darstellbar ist. Der relative Fehler wird dann mit dem folgenden Programm abgeschätzt:

```
// Program:  Improve_T.cpp
#include "abs_relh.hpp" // for rel_divh1()
#include "bnd_arh.hpp"  // for abs_1mx_delh()
#include "bnd_util.hpp" // for Max_bnd_Xi()
#include <iostream>     // for cout

using namespace cxsc;
using namespace std;
```

[5] (3.25) und (3.26) sind die Taylor-Entwicklung von $T(x)$ um den Punkt $x = x_0$.

```
real T_x(const interval& Xi, const real& delx)
{
    interval N,T;
    real delr,deln,eps;
    abs_1mx_delh(Xi,delx,N,deln);    // absl. error: 1-x
    abs_divh1(2*Xi,0,N,deln,T,delr); // absl. error: 2x/(1-x)
    rel_1px_delh(T,delr,N,eps);      // rel.  error: 1+2x/(1-x)
    return eps;   // Returning an upper bound eps of the relative
}                 // error for T(x) = 1 + 2x/(1-x) = (1+x)/(1-x)

int main()
{
    interval X1 = interval(-0.2,+0.2);
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X1,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das Programm `Improve_T.cpp` liefert für das Intervall $[a,b] = [-0.2,+0.2]$ das folgende Ergebnis:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq \varepsilon(T) = 2.868086 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in [-0.2,+0.2]$$

und mit dem etwas breiteren Intervall $[a,b] = [-0.25,+0.25]$ erhält man die schon deutlich größere Schranke:

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq \varepsilon(T) = 3.589747 \cdot 10^{-16} \quad \text{für alle } \widetilde{x} \in [-0.25,+0.25]$$

Anmerkungen:

1. Um möglichst kleine Fehlerschranken zu erhalten, muss man nicht nur optimale Werkzeuge zur automatischen Fehlerabschätzung benutzen, sondern auch in verschiedenen Argumentbereichen $[a,b]$ einen geeigneten Algorithmus wählen.

2. Wählt man dabei zu breite Intervalle $[a,b]$, so wachsen die Fehlerschranken i.a. deutlich an. Um möglichst kleine relative Fehlerschranken zu erhalten, müsste man daher hinreichend viele Teilintervalle $[a,b]$ mit jeweils eigenen Mittelpunkten x_0 benutzen, wodurch die Laufzeit merklich anwachsen würde. Damit wird das folgende Grundprinzip deutlich:

   Zu kleine Fehlerschranken erfodern i.a. einen großen numerischen Aufwand, so dass in solchen Fällen ein sinnvoller Kompromiss gefunden werden muss.

3.3 Anwendungen

Im vorhergehenden Abschnitt 3.2 haben wir für die Terme $1+x, 1-x$ und $x-1$ Funktionen aus dem Modul `bnd_arh` angegeben, mit denen absolute oder relative Gesamtfehlerschranken bei nur hochgenauer Arithmetik optimal berechnet werden konnten. Bei der Auswertung von Taylorpolynomen treten jedoch häufig auch die folgenden Terme auf:

$$1+x^2, \quad 1+\frac{1}{4}x^2, \quad 1-\frac{1}{3}x^2, \quad 1+\frac{2}{3}x^4, \quad \ldots$$

In den folgenden Unterabschnitten zeigen wir, wie auch für diese Terme mit Hilfe der Funktionen aus dem **C-XSC** Modul `bnd_arh` optimale Fehlerschranken berechnet werden können. Dabei wird vorausgesetzt, dass die obigen Terme nur für Maschinenzahlen $\widetilde{x} \in \mathtt{X} = [10^{-8}, 0.1]$ auszuwerten sind. Für alle reellen $x \in \mathtt{X}$ soll also gelten: $\widetilde{x} = x + \Delta_x$, mit: $\Delta_x \leq \Delta(x) = 0$, d.h. $x = \widetilde{x} \in [10^{-8}, 0.1] \cap S(2, 53)$.
Bei der Auswertung von $\widetilde{x} \odot \widetilde{x}$ für $\widetilde{x} \in \mathtt{X}$ kann der absolute Fehler $|(\widetilde{x} \odot \widetilde{x}) - (\widetilde{x} \cdot \widetilde{x})|$ nach Satz 3.1.3 von Seite 51 nicht für alle $\widetilde{x} \in \mathtt{X}$ verschwinden, da der Intervalldurchmesser von `X` größer 0 ist. Die Abschätzung des absoluten Fehlers $|(\widetilde{x} \odot \widetilde{x}) - (\widetilde{x} \cdot \widetilde{x})|$ für $\widetilde{x} \in \mathtt{X}$ muss daher bei vorausgesetzter hochgenauer Arithmetik bei den Grundoperationen mit den entsprechenden Funktionen aus dem Modul `abs_relh` vorgenommen werden.

3.3.1 $T(x) = 1 + x^2$

Das nachfolgende **C-XSC** Programm `book_expl16.cpp` liefert für den Term $T(x) = 1 + x^2$ eine Oberschranke des relativen Fehlers, wenn Addition umd Multiplikation in nur hochgenauer Arithmetik ausgeführt werden.

```
#include "abs_relh.hpp" // Wegen rel_mulh4()
#include "bnd_arh.hpp"  // Wegen rel_1px_epsh()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& epsx)
{
    interval P,T;
    real eps,epsP;
    rel_mulh4(Xi,epsx,Xi,epsx,P,epsP); // epsP: rel. Fehler bzgl. x*x
    rel_1px_epsh(P,epsP,T,eps);      // eps:  rel. Fehler bzgl. 1+x*x
```

```
    return eps; // Rückgabe einer Schranke eps des relativen Fehlers.
}
int main()
{
    interval X;
    string("[1e-8,0.1]") >> X; // [1e-8,0.1] enthalten in X
    real bnd, diam = 1e-5, epsx=0;
    Max_bnd_Xi(T_x,X,epsx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl16.cpp` liefert die Bildschirmausgabe:

```
Relative Fehlerschranke = 2.220447E-016
```

Danach gilt dann für $T(x) = 1 + x^2$ bei nur hochgenauer Arithmetik

$$\left| \frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})} \right| \leq 2.220447 \cdot 10^{-16} \quad \text{für alle } x = \widetilde{x} \in \texttt{X}$$

Anmerkungen:

1. Da durch obige Anweisung `string("[1e-8,0.1]") >> X;` die Einschließung $[10^{-8}, 0.1] \subset \texttt{X}$ garantiert wird, gilt die angegebene relative Fehlerschranke auch für alle $x = \widetilde{x} \in [10^{-8}, 0.1]$.

2. Vorausgesetzt wurde $|x - \widetilde{x}| = |\Delta_x| \leq \Delta(x) = 0$ für alle $x = \widetilde{x} \in \texttt{X}$, d.h. die relative Fehlerschranke $2.220447 \cdot 10^{-16}$ bezieht sich nur auf die Rasterzahlen $\widetilde{x} \in \texttt{X} \cap S(2, 53)$.

3. Mit Hilfe der Funktion `Max_bnd_Xi(...)` aus dem Modul `bnd_util` wird das Intervall `X` mittels `diam = 1e-5` in hinreichend viele Teilintervalle `Xi` zerlegt. Die Funktion `T_x()` berechnet zu jedem Teilintervall `Xi` eine Oberschranke des dort auftretenden relativen Fehlers, und `Max_bnd_Xi(...)` bestimmt dann das Maximum all dieser Oberschranken als den relativen Fehler über dem Gesamtintervall `X`.

4. Im Punkt 4 auf Seite 75 findet man weitere Hinweise zum Einsatz der Funktion `Max_bnd_Xi()`.

5. Das Programm von Seite 84 hat allerdings noch einen entscheidenden Nachteil: Wählt man ein Argumentintervall, das die Null enthält, also z.B. $\texttt{X} = [-1, +1]$, so liefert das letzte Programm für den relativen Fehler die Bildschirmausgabe `<+Infinity>`, obwohl der relative Fehler existiert! Enthält nämlich beim

Aufruf der Funktion `rel_mulh4(Xi,epsx,Xi,epsx,P,epsP)` das Teilintervall `Xi` die Zahl Null, so wird bei der Berechnung des relativen Fehlers `epsP` durch Null dividiert und dadurch das Ergebnis `<+Infinity>` verursacht.

Man umgeht den in Punkt 5 beschriebenen Fehler, indem man für $\widetilde{x} \odot \widetilde{x}$, mit $\widetilde{x} \in$ `Xi` nicht den relativen, sondern nur den **absoluten** Fehler berechnet. Dies erreicht man durch Aufruf der Funktionen:

```
abs_mulh4(Xi,epsx,Xi,epsx,P,delP); // delP: abs. Fehler bzgl. x*x
rel_1px_delh(P,delP,T,eps);
```

wobei letztere mit `eps` eine Oberschranke des gewünschten relativen Fehlers bei Auswertung von $T(x) = 1 + x^2$ liefert. Das verbesserte Programm `book_expl17.cpp`

```
#include "abs_relh.hpp" // Wegen abs_mulh4()
#include "bnd_arh.hpp"  // Wegen rel_1px_delh()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& epsx)
{
    interval P,T;
    real eps,delP;
    abs_mulh4(Xi,epsx,Xi,epsx,P,delP); // delP: abs. Fehler bzgl. x*x
    rel_1px_delh(P,delP,T,eps);      // eps:  rel. Fehler bzgl. 1+x*x
    return eps; // Rückgabe einer Schranke eps des relativen Fehlers.
}

int main()
{
    interval X;
    string("[-1,1]") >> X;
    real bnd, diam = 1e-5, epsx=0;
    Max_bnd_Xi(T_x,X,epsx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

liefert jetzt für `X` $= [-1, +1]$ den korrekten relativen Fehler: `bnd` $= 2.220491 \cdot 10^{-16}$.

3.3.2 $T(x) = 1 + \frac{1}{4} \cdot x^2$

Die Auswertung von $T(x)$ soll wieder für alle $\widetilde{x} \in [10^{-8}, 0.1]$ erfolgen, so dass alle Werte $0.25 \cdot x^2$ stets im normalisierten Zahlenbereich liegen. Die Multiplikation mit 0.25 erfolgt somit rundungsfehlerfrei, was im folgenden Programm durch den Aufruf `abs_mulh4(F,0,P,delP,T,delF)` berücksichtigt wird, bei dem der erste Parameter `F` das Punktintervall $[2^{-2}, 2^{-2}]$ ist. Nach der Bemerkung auf Seite 40 wird daher in der Funktion `abs_mulh4()` der Rundungsfehler bez. der Multiplikation[6] mit 0.25 auf Null gesetzt. Das nachfolgende **C-XSC** Programm `book_expl18.cpp` liefert für $T(x)$ eine Oberschranke des relativen Fehlers, wobei eine nur hochgenaue Arithmetik vorausgesetzt wird.

```
#include "abs_relh.hpp" // Wegen abs_mulh4(), abs_mulh1()
#include "bnd_arh.hpp"  // Wegen rel_1px_delh()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Xi, const real& epsx)
{
    interval P,T, F=interval(0.25);
    real eps,delP,delF;
    abs_mulh4(Xi,epsx,Xi,epsx,P,delP); // delP: abs. Fehler bzgl. x*x
    abs_mulh1(F,0,P,delP,T,delF); // delF: abs. Fehler bzgl. 0.25*x*x
    rel_1px_delh(T,delF,F,eps); // eps : rel. Fehler bzgl. 1+0.25*x*x
    return eps;  // Rückgabe einer Schranke eps des relativen Fehlers.
}
int main()
{
    interval X;
    string("[1e-8,0.1]") >> X; // [1e-8,0.1] enthalten in X
    real bnd, diam = 1e-5, epsx=0;
    Max_bnd_Xi(T_x,X,epsx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl18.cpp` liefert die folgende Bildschirmausgabe:

```
Relative Fehlerschranke = 2.220447E-016
```

Es gelten auch hier wieder die gleichen Anmerkungen wie auf Seite 85.

[6] Aus Laufzeitgründen wird man die Multiplikation mit 0.25 mit der Funktion `times2pown()` ausführen.

3.3.3 $T(x) = 1 + \frac{2}{3} \cdot x^4$

Im letzten Abschnitt musste bei Auswertung von $T(x) = 1+0.25\cdot x^2$ das Maschinenprodukt $\widetilde{x}^2$ mit 0.25 multipliziert werden. Der dabei auftretenden Rundungsfehler war Null, da 0.25 in $S(2,53)$ exakt darstellbar ist, $0.25\cdot\widetilde{x}^2$ für alle $x = \widetilde{x} \in [10^{-8}, 0.1]$ im normalisierten Bereich liegt und $0.25 = 2^{-2}$ als Potenz zur Basis 2 geschrieben werden kann. Bei der jetzigen Multiplikation mit 2/3 ist folgendes zu beachten:

1. Wegen $\frac{2}{3} \notin S(2,53)$ muss dieser Bruch durch ein möglichst enges Maschinenintervall eingeschlossen werden, um unnötige Überschätzungen bei der Fehlerabschätzung zu vermeiden.

2. Der Bruch $\frac{2}{3}$ muss zur nächsten Rasterzahl $\widetilde{a} \in S(2,53)$ gerundet werden, und bei der konkreten Auswertung von $T(x)$ muss die Multiplikation mit $\frac{2}{3}$ ersetzt werden durch die Maschinenmultiplikation mit $\widetilde{a}$.

3. Zur Abschätzung des Auswertefehlers muss der durch die Rundung von $\frac{2}{3}$ nach $\widetilde{a}$ aufgetretene absolute oder relative Fehler bekannt sein.

Diese drei Aufgaben werden gelöst mit Hilfe der im Modul `bnd_util` realisierten Funktion `frac_tools()`. Für den Bruch $\frac{2}{3}$ liefert der Funktionsaufruf

```
frac_tools(2,3,x,a,del,eps23); // interval x;  real a,del,eps23;
```

mit `x` eine optimale Einschließung von $\frac{2}{3}$, mit `a` die zu $\frac{2}{3}$ nächstgelegene Rasterzahl und mit `del` $= 3.700743415417188\ldots\cdot 10^{-17}$ und `eps23` $= 5.5511151231257\ldots\cdot 10^{-17}$ Oberschranken des absoluten und relativen Datenfehlers bez. der Rundung von $\frac{2}{3}$ nach $\widetilde{a} =$ `a` $\in S(2,53)$. Beachten Sie, dass der relative Datenfehler `eps23` deutlich kleiner ist als $\varepsilon(m) = 2^{-53} = 1.1102\ldots\cdot 10^{-16}$, da $\frac{2}{3}$ nicht in der unmittelbaren Umgebung des Mittelpunktes, sondern relativ dicht an einem der beiden Randpunkte von `x` liegt, denn der absolute Datenfehler `del` $= |\frac{2}{3} -$ `a`$|$ ist deutlich kleiner als `diam(x)` $= 1.1102\ldots\cdot 10^{-16}$. Beachten Sie ferner, dass die durch Rundung von $\frac{2}{3}$ zur nächsten Rasterzahl $\widetilde{a} =$ `a` $\in S(2,53)$ bedingten Fehlerschranken `del` und `eps23` völlig unabhängig davon sind, ob die Termauswertung in maximal genauer oder nur in hochgenauer Arithmetik auszuführen ist.

Das nachfolgende **C-XSC** Programm liefert für die Maschinenauswertung des Terms $T(x) = 1 + \frac{2}{3} \cdot x^4$ unter der Voraussetzung $|\widetilde{x} - x| \leq \Delta(x) = 0$, d.h. für alle rundungsfehlerfreien Maschinenzahlen $x = \widetilde{x} \in$ `X` $\supset [10^{-8}, 0.1]$ eine Oberschranke `bnd` des relativen Gesamtfehlers

$$\left|\frac{T(\widetilde{x}) - \widetilde{T}(\widetilde{x})}{T(\widetilde{x})}\right| \leq \texttt{bnd}, \quad \text{für alle } \widetilde{x} \in \texttt{X} \supset [10^{-8}, 0.1]$$

```
#include "abs_relh.hpp"   // Wegen abs_mulh4(), abs_mulh1()
#include "bnd_arh.hpp"    // Wegen rel_1px_delh()
#include "bnd_util.hpp"   // Wegen Max_bnd_Xi(), frac_tools()
#include <iostream>       // Wegen cout

using namespace cxsc;
using namespace std;

interval x;  // Globale Variable x zur optimalen Einschließung von 2/3
real a,del23,eps23;

real T_x(const interval& Xi, const real& epsx)
{
    interval P, P4, T;
    real eps,delP4,delT,delP;
    abs_mulh4(Xi,epsx,Xi,epsx,P,delP); // delP: abs. Fehler bzgl. x*x
    abs_mulh1(P,delP,P,delP,P4,delP4);
                                // delP4: abs. Fehler bzgl. (x*x)*(x*x)
    abs_mulh1(x,del23,P4,delP4,T,delT);
                                   // delT: abs. Fehler bzgl. 2/3 * x^4
    rel_1px_delh(T,delT,P,eps); // eps : rel. Fehler bzgl. 1+2/3 * x^4
    return eps;  // Rückgabe einer Schranke eps des relativen Fehlers.
}

int main()
{
    frac_tools(2,3,x,a,del23,eps23);  // x schließt 2/3 optimal ein;
       // del23 ist relativer Datenfehler bei Rundung von 2/3 zur
       // nächsten Rasterzahl a. del23 und a sind globale Variable
    interval X;
    string("[1e-8,0.1]") >> X; // X: optim. Einschließg. von[1e-8,0.1]
    real bnd, diam = 1e-5, epsx=0;
    Max_bnd_Xi(T_x,X,epsx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Das obige Programm `book_expl19.cpp` liefert die folgende Bildschirmausgabe:

```
Relative Fehlerschranke = 2.220928E-016
```

Anmerkungen:

1. Für die Auswertung des Terms $T(x) = 1 + \frac{2}{3} \cdot x^4$ wurde vorausgesetzt, dass alle auftretenden Grundoperationen $\oplus$ und $\odot$ in nur hochgenauer Arithmetik ausgeführt werden. Die in `T_x()` zur Abschätzung des relativen Auswertefehlers benutzten Funktionen `abs_mulh1()` und `rel_1px_delh()` enthalten in ihrem Funktionsnamen daher den Buchstaben `h`.

2. Bei der Auswertung von $\widetilde{x}^4$ setzen wir voraus, dass zunächst $\widetilde{x} \odot \widetilde{x}$ berechnet und dieses Zwischenergebnis anschließend quadriert wird. Würde man $\widetilde{x}^4$ jedoch als $\widetilde{x} \odot \widetilde{x} \odot \widetilde{x} \odot \widetilde{x}$ mit drei Maschinenmultiplikationen auswerten, so müsste dies bei der Fehlerabschätzung entsprechend berücksichtigt werden! Neben einer längeren Laufzeit hätte man dabei eine größere Fehlerschranke zu akzeptieren.

3. Durch den Aufruf `abs_mulh1(x,del23,P4,delP4,T,delT)` wird dann in `T_x()` mit `delT` der absolute Auswertefehler bei der Multiplikation von $\widetilde{x}^4$ mit der zu $\frac{2}{3}$ nächstgelegenen Rasterzahl `a` berechnet. Bei Auswertung des Terms $T(\widetilde{x})$ wird also vorausgesetzt, dass auf der Maschine mit der zu $\frac{2}{3}$ nächstgelegenen Rasterzahl `a` multipliziert wird. Diese Maschinenzahl `a` und der durch die entsprechenden Rundung bedingte absolute Datenfehler `del23` kann mit Hilfe der Funktion `frac_tools(2,3,x,a,del23,eps23)` berechnet werden. Das Intervall $\mathtt{x} \in IR$ schießt dabei den nicht darstellbaren Bruch $\frac{2}{3}$ optimal ein.

4. Mit dem obigen Funktionsaufruf `Max_bnd_Xi(T_x,X,epsx,diam,bnd)` wird das Maschinenintervall $\mathtt{X} \supset [10^{-8}, 0.1]$ mittels $\mathtt{diam} = 10^{-5}$ in hinreichend viele Teilintervalle unterteilt und für jedes Teilintervall der relative Auswertefehler mit der Funktion `T_x()` einzeln berechnet. Das Maximum dieser Auswertefehler ist dann der relative Auswertefehler für das ganze Intervall `X`. Weitere Einzelheiten zur Arbeitsweise der Funktion `Max_bnd_Xi` findet man in Anmerkung 4 auf Seite 75.

5. Vergrößert man das Maschinenintervall der ungestörten Maschinenzahlen $\widetilde{x} \in \mathtt{X}$ auf $\mathtt{X} \supset [10^{-8}, 0.75]$, so liefert das letzte Programm den größeren relativen Auswertefehler $\mathtt{bnd} = 3.477538 \cdot 10^{-16}$.

Kapitel 4

Auswertefehler

4.1 Problemstellung

Bei der Implementierung der Standardfunktionen oder der speziellen Funktionen der mathematischen Physik spielen Polynome und rationale Funktionen eine zentrale Rolle, weil die Auswertung der Polynome nach dem Hornerschema auf einem Rechner nur kurze Laufzeiten erfordern. In der unmittelbaren Umgebung des Ursprungs können Standardfunktionen oft durch ihre abgebrochenen Taylorreihen vom Grade $N \in \mathbb{N}$ sehr effektiv approximiert werden:

$$P_N(u) = a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \qquad \text{mit:} \tag{4.1}$$
$$u = u(x) = x^k,\ k \in \{1,2,3,4,5,6\}, \quad x \in [a,b] \in I\!I\!R;$$

In der Praxis gilt meist: $a_0 = 1$ und $|a_j| > |a_{j+1}|$. Da die rationalen Koeffizienten a_j i.a. nicht im **IEEE**–Format $S(2,53)$ darstellbar sind, müssen die a_j zur jeweils nächsten Rasterzahl $\widetilde{a}_j$ gerundet werden. Die Polynomauswertung erfolgt also konkret mit den fehlerbehafteten a_j, was bei der notwendigen Fehlerabschätzung entsprechend zu berücksichtigen ist. Beachten Sie bitte, dass wir jetzt mit den Kenntnissen aus den Kapiteln 2 und 3 in der Lage sind, die Fehlerschranken der Multiplikation und Addition in jedem Hornerschritt zu bestimmen. Wir können damit den absoluten oder relativen Fehler abschätzen, wenn das ganze Polynom in hochgenauer Arithmetik nach dem Hornerschema ausgewertet wird. Wegen $a_0 = 1$ ist im letzten Hornerschritt als letzte Operation 1 zu addieren, so dass wir zur Fehlerabschätzung bei nur hochgenauer Arithmetik die im Modul `bnd_arh` implementierte Funktion `rel_1px_epsh()` zur Optimierung des relativen Auswertefehlers benutzen können, vgl. dazu Seite 69.

Wir betrachten jetzt eine neue Aufgabenstellung: Eine stetig differenzierbare, reelle Funktion soll in einer nicht zu breiten Umgebung der Maschinenzahl $x_0 \neq 0$ durch folgendes Polynom approximiert werden:

$$P_N(u) = a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \qquad \text{mit:} \tag{4.2}$$
$$u = u(x - x_0) = (x - x_0)^k, \quad k \in \{1, 2, 3, 4, 5, 6\}, \quad x \in [a, b] \in I\!I\!R;$$

Die mit einem Computeralgebrasystem berechneten Polynomkoeffizienten liegen zunächst als Dezimalzahlen[1] vor und müssen deshalb zur jeweils nächsten Rasterzahl $a_j \in S(2, 53)$ gerundet werden. Wir betrachten damit die Maschinenzahlen a_j jetzt als die **exakten** Polynomkoeffizienten, nehmen dabei jedoch in Kauf, dass sich der ursprüngliche Approximationsfehler durch die notwendigen Rundungen erfahrungsgemäß etwa um den Faktor 3 vergrößert. Dies ist jedoch kein wirklicher Nachteil, wenn man den Approximationsfehler vorher durch einen geeigneten Polynomgrad N hinreichend klein gewählt hat. Der eigentliche Vorteil dieser Vorgehensweise besteht nun in einer deutlichen Reduzierung des Auswertefehlers, wenn man die gerundeten Polynomkoeffizienten als exakte Werte betrachtet. Wie eine gesicherte Schranke des vergrößerten Approximationsfehlers zu berechnen ist, wird ausführlich in Kapitel 7 beschrieben.

Um möglichst kleine Auswertefehler zu erhalten, wird man das Approximationsintervall, d.h. also das Intervall der exakten Polynomargumente, in hinreichend viele Teilintervalle unterteilen und in jedem Teilintervall den Auswertefehler einzeln berechnen. Das Maximum dieser Auswertefehler ist dann eine optimierte Oberschranke für das ganze Approximationsintervall. Diese Vorgehensweise haben wir bereits in Kapitel 3 auf Seite 75 kennen gelernt.

Noch eine Anmerkung zum Polynom $P_N(x-x_0)$ in (4.2). Man könnte auf die Idee kommen, den Term rechts in (4.2) auszumultiplizieren, um dadurch ein Polynom nur in x zu erhalten. Die Erfahrung zeigt jedoch, dass der Auswertefehler dieses neuen Polynoms bei nicht zu breiten Argumentintervallen deutlich größer ausfällt als beim Polynom $P_N(x - x_0)$, zumal die Differenzen $(\widetilde{x} - x_0)$ auf der Maschine bei vielen Anwendungen rundungsfehlerfrei berechnet werden können, vgl. dazu das Beispiel 1 auf Seite 108. Die Polynomauswertefehler der Polynome in (4.2) werden also minimal, wenn

1. die Polynomkoeffizienten $a_j \in S(2, 53)$ als **exakt** betrachtet werden
2. der Polynomterm rechts in (4.2) mit $u = (x - x_0)^k$ nicht ausmultipliziert wird
3. das Argumentintervall $[a, b]$ in hinreichend viele Teilintervalle zerlegt wird
4. die Differenzen $(\widetilde{x} - x_0)$ auf der Maschine exakt berechnet werden können
5. $P_N(x - x_0)$ nach dem Hornerschema ausgewertet wird.

[1] Eine Präzision von 18 Dezimalziffern wird i.a. ausreichend sein, weil die Präzision des **IEEE**–double Formats durch etwa 16 Dezimalziffern gegeben ist.

Die Berechnung von $(x - x_0)^k$ ist für $k = 1, 2, 3$ eindeutig; für $k = 4, 5, 6$ ist die Potenz $(x - x_0)^k \equiv d^k$ wie folgt auszuwerten:

$$k = 4: \qquad d^4 := (d \odot d) \odot (d \odot d)$$

$$k = 5: \qquad d^5 := (d \odot d) \odot (d \odot d) \odot d$$

$$k = 6: \qquad d^6 := (d \odot d) \odot (d \odot d) \odot (d \odot d)$$

Im Spezialfall $x_0 = 0$ und $k = 1$, d.h. $u = x$ wird die Auswertung des Polynoms $P_3(x) = 1 + \frac{1}{2} \cdot x - \frac{1}{4} \cdot x^2 + \frac{1}{8} \cdot x^3$ für $x = \frac{1}{8}$ nach dem Hornerschema im folgenden Programm `book_expl20.cpp` demonstriert:

```
#include <iostream>    // Wegen cout
#include <rvector.hpp> // Wegen rvector a(0,N);
using namespace cxsc;  using namespace std;
int main()
{
    int N=3; // Polynomgrad 3
    rvector a(0,N); // Polynomkoeffizienten a[j], j=0,1,2,3;
    a[0]=1;  a[1]=0.5;  a[2]=-0.25;  a[3]=0.125;
    real x=0.125, y = a[Ub(a)]; // y: Startwert für Hornerschema
    // Horner-Schema:
    for (int j=Ub(a)-1; j>=Lb(a); j--)
    {
        y = y*x+a[j];
    } // y: Maschinen-Näherung für P_3(x);
    cout << RndNext << SetPrecision(16,16)
         << "Polynom-Näherung = " << y << endl;
}
```

Die Bildschirmausgabe `Polynom-Näherung = 1.0588378906250000` liefert jetzt sogar den exakten Polynomwert $P_3(0.125) = 1.058837890625$; offensichtlich wird das Hornerschema in maximal genauer Arithmetik ausgeführt, wobei alle auftretenden Additionen und Multiplikationen vermutlich rundungsfehlerfrei erfolgen. Beachten Sie bitte, dass in diesem Beispiel alle Polynomkoeffizienten $a[j]$ exakt gespeichert werden können.

Spezielle Funktionen der mathematischen Physik können in der Umgebung einer Maschinenzahl $\widetilde{x}_0$ oftmals durch rationale Funktionen

$$R(u) := \frac{a_0 + a_1 \cdot u^1 + \ldots + a_N \cdot u^N}{b_0 + b_1 \cdot u^1 + \ldots + b_M \cdot u^M}, \quad u = (x - x_0)^k \tag{4.3}$$

sehr effektiv approximiert werden, wobei das Nennerpolynom nicht verschwinden darf. Für eine gesicherte Fehlerabschätzung ist dann der absolute oder relative Auswertefehler dieser Funktion $R(u)$ zu berechnen. Auch diese Aufgabe kann mit den uns zur Verfügung stehenden Mitteln gelöst werden, da wir ja grundsätzlich schon wissen, wie man die entsprechenden Auswertefehler der Polynome abschätzen kann. Für eine optimale Fehlerabschätzung sind dabei folgende Punkte zu beachten:

1. Alle Polynomkoeffizienten a_j, b_j sind als **exakte** Maschinenzahlen aufzufassen, die durch Rundung der dezimalen Näherungen eines Computeralgebrasystems zur jeweils nächsten Maschinenzahl entstanden sind.

2. Die Maschinenzahl $x_0 \in S(2, 53)$ sollte so gewählt werden, dass alle Differenzen $(x - x_0)$ auf der Maschine rundungsfehlerfrei berechnet werden können.

3. Auch jetzt sollten die Potenzen $(x - x_0)^k$ **nicht** ausmultipliziert werden, vgl. dazu die entsprechenden Bemerkungen auf Seite 92.

4. Im Falle $b_0 \neq 0$ kann man in (4.3) durch geeignetes Erweitern stets $b_0 = 1$ erzwingen. Dadurch lässt sich dann im letzten Hornerschritt bei Addition der 1 für eine optimale Fehlerabschätzung die Funktion `rel_1px_epsh` aus dem Modul `bnd_arh` benutzen, wenn eine relative Fehlerschranke zu berechnen ist.

5. Wie bei den Polynomen sollte auch jetzt ein zu breites Argumentintervall `X` in hinreichend viele Teilintervalle `Xi` zerlegt werden und in jedem `Xi` die Fehlerabschätzung einzeln vorgenommen werden. In jedem `Xi` sollte dann aber die Fehlerschranke für das Zähler- **und** Nennerpolynom **und** für den anschließenden Maschinenquotienten beider Polynomwerte berechnet werden! Bestimmt man nämlich die Fehlerschranken über dem ganzen Intervall `X` für jedes Polynom einzeln[2] und anschließend die Fehlerschranke des Polynomquotienten, so erhält man i.a. eine erhebliche Überschätzung der tatsächlichen Fehlerschranke.

Aus dem letzten Punkt ergibt sich damit die folgende Regel:

> Ist für einen gegebenen Term $T(x)$ der Auswertefehler über einem zu breiten Intervall `X` abzuschätzen, so sollte `X` in hinreichend viele Teilintervalle `Xi` zerlegt werden. In jedem `Xi` ist dann die Fehlerabschätzung für den **ganzen** Term vorzunehmen. Eine Aufspaltung von $T(x)$ in mehrere Einzelterme und die gesonderte Fehlerabschätzung für jeden Einzelterm ergibt i.a. eine deutliche Überschätzung des tatsächlichen Fehlers!

[2] natürlich auch mit Hilfe einer geeigneten Intervallunterteilung!

4.2 Polynome mit rationalen Koeffizienten

Gegeben sei für $x \in [a,b] \in I\!I\!R$ das folgende Polynom

$$P_N(u) = 1 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \tag{4.4}$$
$$u = u(x) = x^k, \quad k \in \{1,2,3,4,5,6\}, \quad x \in [a,b] \in I\!I\!R;$$

mit rationalen Koeffizienten a_j, wie sie z.B. bei Taylorpolynomen meist realisiert sind. Wir setzen dabei voraus, dass alle a_j zur jeweils nächsten Rasterzahl $\widetilde{a}_j$ des **IEEE**–double–Formats gerundet werden. Bezeichnet man dann mit $\widetilde{P}_N(\widetilde{u})$ den mit diesen gerundeten Koeffizienten auf der Maschine nach dem Hornerschema ausgewerteten Polynomwert zum Maschinenargument $\widetilde{x}$, so ist der relative Auswertefehler definiert durch:

$$\varepsilon_P := \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)}, \qquad u = x^k, \quad x \in [a,b] \in I\!I\!R;$$

Wir interpretieren dabei das Intervall $[a,b]$ als den exakten Wertebereich einer Funktion mit den exakten Funktionswerten x und nehmen weiter an, dass diese x auf dem Computer fehlerhaft mit dem jeweiligen Maschinenergebnis $\widetilde{x} \in S(2,53)$ berechnet wurden. Für alle $x \in [a,b]$ werden dabei die absoluten und relativen Fehlerschranken $\Delta(x)$ und $\varepsilon(x)$ als bekannt angesehen[3]:

$$|\widetilde{x} - x| \le \Delta(x), \qquad |\widetilde{x} - x| \le |x| \cdot \varepsilon(x), \quad \text{für alle } x \in [a,b] \in I\!I\!R$$

Setzt man $\Delta(x) = 0$ oder $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler ε_P ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a,b]$.

4.2.1 Hochgenaue Arithmetik

Bei hochgenauer Arithmetik liefert das Programm mit dem nachfolgenden Quelltext `Poly1_relh.cpp` mit der `real`-Variablen `bnd` für das Polynom in (4.4) eine Oberschranke für den Auswertefehler

$$\left| \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)} \right| \le \varepsilon(P) = \texttt{bnd}, \qquad u = x^k, \quad x \in [a,b] \in I\!I\!R; \tag{4.5}$$

Nach dem Programmstart muss der Anwender die folgenden Werte eingeben:

- Polynomgrad N
- Exponent k im Ausdruck $u = x^k$, mit $1 \le k \le 6$;

[3] $\Delta(x)$ und $\varepsilon(x)$ sind Konstanten und keine von x abhängigen Größen!

- integer-Zahlen für Zähler und Nenner der Koeffizienten $a_j,\ j = 1, 2, \ldots, N$;
- Das Intervall $[a, b]$ der exakten Polynomargumente x.
- Die relative Fehlerschranke `epsx` $= \varepsilon(x)$ der gestörten Argumente
- Die gewünschte Mantissenbreite zur Intervallzerlegung; `diam` $= 10^{-5}$ ist meist eine gute Wahl.

```
// Programm:  Poly1_relh.cpp Zur Abschätzung des relativen
//            Auswertefehlers bei hochgenauer Arithmetik
#include "bnd_util.hpp"   // Wegen Horn_relh1(),Poly_Dat_rel(), ...
#include <iostream>       // Wegen cout
using namespace cxsc;
using namespace std;
int main()
{ // Polynomauswertung mit Hornerschema;  Gesucht: Rel. Auswertefehler
  // P(x) = 1 + a[1]*u^1 + ... + a[N]*u^N,  u(x)=x^k;
    int N,k;     // N: Polynomgrad;  k: u(x)=x^k, mit  1<= k <=6;
    interval X;  // Intervall der exakten Polynom-Argumente x
    real bnd, diam, epsx;
    Polynomgrad(N,k);      // Interaktive Eingabe von  N und k
    intvector Za(N),Ne(N); // Lb(Za)=Lb(Ne)=1, Ub(Za)=Ub(Ne)=N;
    Koeff_Rat(Za,Ne);  // Koeffizienten-Eingabe initialisiert Za,Ne;
    while(1) // Endlosschleife
    {
        Poly_Dat_rel(X,epsx,diam);  // Eingabe von X,epsx,diam;
            // epsx: relative Fehlerschranke der gestörten Argumente
            // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
        bnd = Horn_relh1(Za,Ne,X,epsx,k,diam); // bnd: relat. Fehler-
        // schranke bei nur hochgenauer Arithmetik beim Hornerschema
        cout << RndUp
             << "Relative Fehlerschranke bei hochgenauer Arithmetik ="
             << bnd << endl << endl;
    }
}
```

Innerhalb der `while`-Schleife kann man für das gleiche Polynom den relativen Auswertefehler mit anderen Eingabewerten für $[a, b], \varepsilon(x)$ und für die Mantissenbreite der Teilintervalle neu berechnen, ohne die gleichen Polynomkoeffizienten jedesmal neu eingeben zu müssen.

Anmerkungen:

- Es muss gelten $a_0 = 1$. Die exakten, rationalen Koeffizienten a_j müssen zur jeweils nächsten Rasterzahl gerundet werden. Nach Eingabe dieser rationalen Koeffizienten erfolgt deren Rundung mit Hilfe der Funktion `Koeff_tools`, die innerhalb der Funktion `Horn_relh1(Za,Ne,X,epsx,k,diam)` aufgerufen wird.
- Das Polynom ist mit diesen i.a. fehlerbehafteten Koeffizienten auf dem Rechner nach dem Hornerschema auszuwerten, und das Programm `Poly1_relh` berechnet eine garantierte Oberschranke des dabei auftretenden relativen Auswertefehlers.
- Innerhalb der Funktion `Horn_relh1(Za,Ne,X,epsx,k,diam)` wird das Intervall `X` der exakten Polynomargumente x in hinreichend viele Teilintervalle `Xi` zerlegt, und durch Aufruf der Funktion `Horn_relh1_Xi` der relative Auswertefehler für jedes dieser `Xi` einzeln berechnet. Das Maximum dieser Fehler ist dann der gesuchte relative Auswertefehler für das ganze Argumentintervall `X`, wenn das Polynom mit den i.a. fehlerbehafteten Koeffizienten $\widetilde{a}_j$ über dem Intervall `X` nach dem Hornerschema ausgewertet wird.

Für weitere Überlegungen ist der Quelltext der Funktion `Horn_relh1_Xi()` aus dem Modul `bnd_util` nachfolgend angegeben:

```
real Horn_relh1_Xi(const interval& Xi, const real& epsx,
                   const ivector& ia, const rvector& del,const int& k)
// Berechnung der relativen Fehlerschranke über einem Teilintervall
// Xi, welches die exakten Polynom-Argumente enthält.
// Vorausgesetzt wird eine nur hochgenauer Arithmetik, mit u(x) = x^k.
// epsx ist die rel. Fehlerschranke der gestörten Polynomargumente.
// ia importiert die optimalen Einschließungen der exakten rationalen
// Polynom-Koeffizienten.
// del importiert die abs. Fehlerschranken bezüglich der Rundung der
// exakten Polynomkoeffizienten zur jeweils nächsten IEEE-Rasterzahl.
// Das Zeichen '1' im Funktionsnamen bedeutet, dass der Polynom-
// koeffizient a[0] == 1 sein muss!!
{
    real del1,del2,delu;
    interval u,h1,h2;
    delu = Sup( abs(Xi) );
    delu = mulu(delu,epsx);

// Auswertung von u(x) = x^k:
```

```
switch (k) {
    case 1:  // u(x) = x^1
        u = Xi;
        break;
    case 2:  // u(x) = x^2
        abs_mulh4(Xi,epsx,Xi,epsx,u,delu);
        break;
    case 3:
        abs_mulh4(Xi,epsx,Xi,epsx,h1,del1);
                            // h1: Einschließg. für Xi*Xi
        abs_mulh2(h1,del1,Xi,epsx,u,delu);
                            // u: Einschg. für (Xi*Xi)*Xi
        break;
    case 4:  // u(x) = x^4
        abs_mulh4(Xi,epsx,Xi,epsx,h1,del1);
                            // h1: Einschließg. für Xi*Xi
        abs_mulh1(h1,del1,h1,del1,u,delu);
                            // u:  Einschlg. für (Xi*Xi)^2
        break;
    case 5:  // u(x) = x^5
        abs_mulh4(Xi,epsx,Xi,epsx,h1,del1);
                            // h1: Einschließg. für Xi*Xi
        abs_mulh1(h1,del1,h1,del1,h2,del2);
                            // h2: Einschlg. für (Xi*Xi)^2
        abs_mulh2(h2,del2,Xi,epsx,u,delu);
                            // u:  Einschlg. für Xi^5
        break;
    case 6:  // u(x) = x^6
        abs_mulh4(Xi,epsx,Xi,epsx,h1,del1);
                            // h1: Einschließg. für Xi*Xi
        abs_mulh1(h1,del1,h1,del1,h2,del2);
                            // h2: Einschlg. für (Xi*Xi)^2
        abs_mulh1(h2,del2,h1,del1,u,delu);
                            // u:  Einschlg. für Xi^6
        break;
    default:
        cerr << "Horn_relh1_Xi(): Unzulässige Potenz x^k"
             << endl;
        exit(1);
} // switch
```

```
    // u ist Einschließung von Xi^k;
    // delu ist die zugehörige absolute Fehlerschranke;
    h1 = ia[Ub(ia)];     // h1: Startwert für Hornerschema
    del1 = del[Ub(ia)];  // del1: absolute Fehlerschranke
    int m = Ub(ia)-1;
    for (int j=m; j>=0; j--)
    {
        abs_mulh1(h1,del1,u,delu,h2,del2);      // h2 = y*u
        if (j==0) rel_1px_delh(h2,del2,h1,del1);
        else abs_addh1(ia[j],del[j],h2,del2,h1,del1);
                                            // h1 = y*u + a[j]
    } // del1: Relativer Fehler des Hornerschemas
      //       wegen rel_1px_delh()
    return del1;   // Rückgabe der relativen Fehlerschranke
} // Horn_relh1_Xi
```

Anmerkungen:

- In der `switch(k)` Anweisung wird mit dem Intervall `u` eine Einschließung für x^k, mit $x \in$ `Xi` berechnet, und `delu` liefert den absoluten Fehler, wenn x^k nach Seite 93 ausgewertet wird. Mit `epsx` kann der relative Fehler der gestörten Polynomargumente für alle $x \in$ `X` angegeben werden. Dieser relative Fehler wird dann durch `delu` automatisch mitberücksichtigt.

- In obiger `for`–Schleife wird der relative Auswertefehler bei Anwendung des Hornerschemas abgeschätzt. Beachten Sie bitte, dass erst bei der Addition von $a_0 = 1$ im letzten Hornerschritt der relative Fehler berechnet wird, während für alle anderen Zwischenergebnisse nur der absolute Fehler abgeschätzt werden darf, um die Division durch Null bei einem verschwindenden Zwischenergebnis zu vermeiden; vgl. dazu auch die entsprechenden Anmerkungen von Seite 85.

> Soll für einen Term der relative Auswertefehler abgeschätzt werden, so darf man für alle Zwischenergebnisse nur die jeweils **absoluten** Fehler berechnen, um unnötige Divisionen durch Null zu vermeiden. Erst bei der im Term zuletzt auszuführenden Grundoperation ist dann der gesuchte relative Auswertefehler zu bestimmen!

Beachten Sie bitte, dass in der `for`–Schleife die letzte Grundoperation beim Hornerschema die Addition von $a_0 = 1$ ist und dass erst jetzt der entsprechende relative Auswertefehler durch den Aufruf `rel_1px_delh(h2,del2,h1,del1)` berechnet wird.

4.2.2 Beispiele

In den folgenden Beispielen betrachten wir die in (4.4) definierten Polynome

$$P_N(u) = 1 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \tag{4.6}$$
$$u = u(x) = x^k, \quad k \in \{1,2,3,4,5,6\}, \quad x \in [a,b] \in I\!I\!R;$$

mit rationalen Koeffizienten a_j, die alle vor der Polynomauswertung zur nächsten Rasterzahl $\widetilde{a}_j$ zu runden sind. Mit dem Programm `Poly1_relh` von Seite 96 kann dann eine Oberschranke $\varepsilon(P)$ des relativen Auswertefehlers

$$\left| \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)} \right| \leq \varepsilon(P) = \texttt{bnd}, \qquad u = x^k, \quad x \in [a,b] \in I\!I\!R \tag{4.7}$$

bei hochgenauer Arithmetik berechnet werden.

Beispiel 1 (Polynome mit rationalen Koeffizienten)

$$P_3(x) = 1 + \frac{1}{3} \cdot x^1 + \frac{1}{5} \cdot x^2 + \frac{1}{7} \cdot x^3$$

In folgender Tabelle sind die mit dem Programm `Poly1_relh` berechneten relativen Auswertefehler $\varepsilon(P)$ zusammengestellt. `X` $= [a,b]$ ist dabei das Intervall der exakten Polynomargumente x, und $\varepsilon(x)$ ist die vorgegebene relative Fehlerschranke der gestörten Argumente $\widetilde{x}$, mit: $|x - \widetilde{x}| \leq |x| \cdot \varepsilon(x)$ für alle $x \in [a,b] =$ `X`. Setzt man $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler $\varepsilon(P)$ ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a,b]$. Die Eingangsgröße `diam` legt die relative Mantissenbreite der Teilintervalle `Xi` fest, in die `X` $= [a,b]$ zur Optimierung der relativen Fehlerschranke $\varepsilon(P)$ zerlegt wird. Mit `diam` $= 10^{-5}$ wählt man in den meisten Fällen eine hinreichend feine Unterteilung von $[a,b]$.

Beachten Sie bitte, dass $P_3(x)$ bei $x_0 = -1.99233\ldots$ eine Nullstelle besitzt und dass $\varepsilon(P)$ umso stärker anwächst, je dichter das Argumentintervall $[a,b]$ an dieser Nullstelle liegt. Wählt man $[a,b] = [-2, -10^{-18}]$, so liegt x_0 im Innern des Argumentintervalls und in einem der Teilintervalle `Xi` wird die Einschließung der zugehörigen Polynomwerte die Zahl Null enthalten, so dass der relative Auswertefehler dann nicht mehr berechnet werden kann. Das Programm `Poly1_relh` liefert in diesem Fall das Ergebnis `<+Infinity>`.

$[a, b]$	$\varepsilon(x)$	diam	$\varepsilon(P)$
$[10^{-18}, 0.007]$	0	10^{-5}	$2.226979 \cdot 10^{-16}$
$[-0.007, -10^{-18}]$	0	10^{-5}	$1.124499 \cdot 10^{-16}$
$[10^{-18}, 0.1]$	0	10^{-5}	$2.325037 \cdot 10^{-16}$
$[-0.1, -10^{-18}]$	0	10^{-5}	$1.319806 \cdot 10^{-16}$
$[10^{-18}, 1]$	0	10^{-5}	$3.925576 \cdot 10^{-16}$
$[-1, -10^{-18}]$	0	10^{-5}	$3.593639 \cdot 10^{-16}$
$[10^{-18}, 1.5]$	0	10^{-5}	$5.997092 \cdot 10^{-16}$
$[-1.5, -10^{-18}]$	0	10^{-5}	$9.191122 \cdot 10^{-16}$
$[10^{-18}, 1.7]$	0	10^{-5}	$6.565245 \cdot 10^{-16}$
$[10^{-18}, 1.7]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$8.058389 \cdot 10^{-16}$
$[-1.7, -10^{-18}]$	0	10^{-5}	$1.914533 \cdot 10^{-15}$
$[-1.7, -10^{-18}]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$2.458473 \cdot 10^{-15}$
$[10^{-18}, 1.9]$	0	10^{-5}	$6.742688 \cdot 10^{-16}$
$[-1.9, -10^{-18}]$	0	10^{-5}	$7.354366 \cdot 10^{-15}$
$[10^{-18}, 1.99]$	0	10^{-5}	$6.742688 \cdot 10^{-16}$
$[-1.99, -10^{-18}]$	0	10^{-5}	$3.152899 \cdot 10^{-13}$
$[10^{-18}, 2]$	0	10^{-5}	$6.742688 \cdot 10^{-16}$
$[-2, -10^{-18}]$	0	10^{-5}	`<+Infinity>`
$[2, 10^{+20}]$	0	10^{-5}	$1.387843 \cdot 10^{-15}$

Tabelle 4.1: Relativer Auswertefehler $\varepsilon(P)$ für $x \in [a, b]$

Man erkennt, dass die relative Fehlerschranke $\varepsilon(P)$ anwächst, wenn $P_3(x)$ für immer größere Argumente x auszuwerten ist. Dies ist leicht zu verstehen, da bei größeren x-Argumenten die Auswertefehler im Wesentlichen durch die Summanden $\frac{1}{5} \cdot x^2$ und $\frac{1}{7} \cdot x^3$ bestimmt werden, während für $|x| \to 0$ der Auswertefehler praktisch nur durch die Summe $1 + \frac{1}{3} \cdot x$ verursacht wird und der dabei entstehende absolute Fehler für $|x| \to 0$ beliebig klein wird. Dass der entsprechende relative Auswertefehler nicht ebenfalls gegen Null geht, sondern dem Wert $\varepsilon(h) = 2^{-52} = 2.2204\ldots \cdot 10^{-16}$ beliebig nahe kommt, ist durch die vorausgesetzte hochgenaue Arithmetik begründet.

Beispiel 2 (Polynome mit rationalen Koeffizienten)

$$P_3(u) = 1 + \frac{1}{3}\cdot u^1 + \frac{1}{5}\cdot u^2 + \frac{1}{7}\cdot u^3, \quad u(x) = x^2, \text{ d.h. } k = 2, \quad x \in [a,b];$$

In folgender Tabelle sind die mit dem Programm `Poly1_relh` berechneten relativen Auswertefehler $\varepsilon(P)$ zusammengestellt. `X` $= [a,b]$ ist dabei das Intervall der exakten Polynomargumente x, und $\varepsilon(x)$ ist die vorgegebene relative Fehlerschranke der gestörten Argumente $\widetilde{x}$, mit: $|x-\widetilde{x}| \le |x|\cdot\varepsilon(x)$ für alle $x \in [a,b] =$ `X`. Setzt man $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler $\varepsilon(P)$ ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a,b]$. Die Eingangsgröße `diam` legt die relative Mantissenbreite der Teilintervalle `Xi` fest, in die `X` $= [a,b]$ zur Optimierung der relativen Fehlerschranke $\varepsilon(P)$ zerlegt wird. Mit `diam` $= 10^{-5}$ wählt man in den meisten Fällen eine hinreichend feine Unterteilung von $[a,b]$. Bei der Eingabe der Polynomdaten ist jetzt der Polynomgrad $N = 3$ und der Exponent $k = 2$ einzugeben.

$[a,b]$	$\varepsilon(x)$	diam	$\varepsilon(P)$
$[10^{-18}, 10^{-5}]$	0	10^{-5}	$2.220447 \cdot 10^{-16}$
$[10^{-18}, 10^{-5}]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$2.220447 \cdot 10^{-16}$
$[10^{-18}, 0.007]$	0	10^{-5}	$2.220528 \cdot 10^{-16}$
$[10^{-18}, 0.007]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$2.220564 \cdot 10^{-16}$
$[10^{-18}, 0.1]$	0	10^{-5}	$2.237279 \cdot 10^{-16}$
$[10^{-18}, 0.1]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$2.244745 \cdot 10^{-16}$
$[10^{-18}, 1]$	0	10^{-5}	$5.314636 \cdot 10^{-16}$
$[10^{-18}, 1]$	$1.110224 \cdot 10^{-16}$	10^{-6}	$6.853746 \cdot 10^{-16}$
$[10^{-18}, 1]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$6.853832 \cdot 10^{-16}$
$[10^{-18}, 1]$	$1.110224 \cdot 10^{-16}$	10^{-3}	$6.863237 \cdot 10^{-16}$
$[10^{-18}, 1]$	$1.110224 \cdot 10^{-16}$	10^{-2}	$6.948560 \cdot 10^{-16}$
$[10^{-18}, 20]$	0	10^{-5}	$1.941461 \cdot 10^{-15}$
$[10^{-18}, 20]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$2.606855 \cdot 10^{-15}$

Tabelle 4.2: Relativer Auswertefehler $\varepsilon(P)$ für $x \in [a,b]$

Man erkennt die wachsende Überschätzung des tatsächlichen relativen Fehlers $\varepsilon(P)$, wenn durch zu große `diam`-Werte die Zerlegung von $[a,b]$ zu grob ausfällt. Wegen $u(x) = x^2$ werden die im Beispiel 1 beschriebenen Effekte für $|x| \to 0$ und $|x| \to +\infty$ hier noch weiter verstärkt.

Beispiel 3 (Polynome mit rationalen Koeffizienten)

$$P_3(u) = 1 - \frac{1}{3} \cdot u^1 + \frac{1}{5} \cdot u^2 - \frac{1}{7} \cdot u^3, \quad u(x) = x^2, \text{ d.h. } k = 2, \quad x \in [a, b];$$

In folgender Tabelle sind die mit dem Programm `Poly1_relh` berechneten relativen Auswertefehler $\varepsilon(P)$ zusammengestellt. `X` $= [a, b]$ ist dabei das Intervall der exakten Polynomargumente x, und $\varepsilon(x)$ ist die vorgegebene relative Fehlerschranke der gestörten Argumente $\widetilde{x}$, mit: $|x - \widetilde{x}| \leq |x| \cdot \varepsilon(x)$ für alle $x \in [a, b] =$ `X`. Setzt man $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler $\varepsilon(P)$ ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a, b]$. Die Eingangsgröße `diam` legt die relative Mantissenbreite der Teilintervalle `Xi` fest, in die `X` $= [a, b]$ zur Optimierung der relativen Fehlerschranke $\varepsilon(P)$ zerlegt wird. Mit `diam` $= 10^{-5}$ wählt man in den meisten Fällen eine hinreichend feine Unterteilung von $[a, b]$. Bei der Eingabe der Polynomdaten ist jetzt der Polynomgrad $N = 3$ und der Exponent $k = 2$ einzugeben.

$[a, b]$	$\varepsilon(x)$	diam	$\varepsilon(P)$
$[10^{-18}, 10^{-5}]$	0	10^{-5}	$1.110224 \cdot 10^{-16}$
$[10^{-18}, 10^{-5}]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$1.110224 \cdot 10^{-16}$
$[10^{-18}, 0.007]$	0	10^{-5}	$1.110360 \cdot 10^{-16}$
$[10^{-18}, 0.007]$	$1.110224 \cdot 10^{-16}$	10^{-5}	$1.110396 \cdot 10^{-16}$
$[10^{-18}, 0.3]$	0	10^{-5}	$1.367077 \cdot 10^{-16}$
$[10^{-18}, 0.3]$	$1.110224 \cdot 10^{-16}$	10^{-6}	$1.435881 \cdot 10^{-16}$
$[10^{-18}, 0.5]$	0	10^{-5}	$1.735324 \cdot 10^{-16}$
$[10^{-18}, 0.5]$	$1.110224 \cdot 10^{-16}$	10^{-6}	$1.921280 \cdot 10^{-16}$

Tabelle 4.3: Relativer Auswertefehler $\varepsilon(P)$ für $x \in [a, b]$

Vergleicht man die jeweils ersten Zeilen der Tabellen 4.3 und 4.2, so erkennt man, dass in diesem Beispiel der relative Fehler $\varepsilon(P) = 1.110224 \cdot 10^{-16}$ um den Faktor 2 kleiner ausfällt als im letzten Beispiel. Dies wird begründet durch den negativen Summanden $-\frac{1}{3} \cdot u^1$ und durch die 7. Zeile in Tabelle 3.1 auf Seite 59. Danach ist bei nur hochgenauer Arithmetik der absolute Rundungsfehler für $1 \oplus y$, mit $y \in (-2^{-4}, 0]$ nicht größer als $2^{-53} = 1.110223\ldots \cdot 10^{-16}$. Für $x \in [10^{-18}, 10^{-5}]$ ist mit $y = -\frac{1}{3} \cdot u^1 + \frac{1}{5} \cdot u^2 - \frac{1}{7} \cdot u^3$ die Bedingung $y \in (-2^{-4}, 0]$ offensichtlich erfüllt, so dass der absolute Rundungsfehler tatsächlich nach Tabelle 3.1 durch 2^{-53} gegeben ist. Beachten Sie bitte, dass auch der bei der Auswertung von y auftretende

fortgepflanzte absolute Datenfehler bei der Berechnung von $\varepsilon(P)$ berücksichtigt wird, aber wegen der sehr kleinen $x \in [10^{-18}, 10^{-5}]$ numerisch nicht in Erscheinung tritt.

4.3 Polynome mit binären Koeffizienten

In diesem Abschnitt soll der relative Auswertefehler folgender Polynome

$$P_N(u) = a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \qquad \text{mit:} \tag{4.8}$$
$$u = u(x - x_0) = (x - x_0)^k, \quad k \in \{1, 2, 3, 4, 5, 6\}, \quad x \in [a, b]$$

berechnet werden. Wir machen dabei die folgenden Voraussetzungen:

1. Das obige Polynom wird in hochgenauer Arithmetik nach dem Hornerschema ausgewertet.
2. x_0 ist eine Maschinenzahl des **IEEE**-double-Formats $S(2, 53)$.
3. Alle $a_j \in S(2, 53)$ werden als rundungsfehlerfreie Maschinenzahlen angesehen, die z.B. durch Rundung vorgegebener dezimaler Näherungswerte[4] ins Raster $S(2, 53)$ entstanden sein können.

Bezeichnet man mit $\widetilde{u}$ die auf der Maschine ausgewertete Potenz $(\widetilde{x} - x_0)^k$, so gilt: $\widetilde{u} = (\widetilde{x} \ominus x_0) \odot (\widetilde{x} \ominus x_0) \odot \ldots$, wobei die von k abhängige Auswertevorschrift auf Seite 93 angegeben ist. Bedeutet dann $\widetilde{P}_N(\widetilde{u})$ den für das Maschinenargument $\widetilde{x}$ auf der Maschine ausgewerteten Polynomwert, so gilt für den relativen Auswertefehler:

$$\varepsilon_P := \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)}, \qquad u = (x - x_0)^k, \quad x \in [a, b] \in I\!I\!R;$$

Wir interpretieren dabei das Intervall $[a, b]$ als den exakten Wertebereich einer Funktion mit den exakten Funktionswerten x und nehmen weiter an, dass diese x auf dem Computer fehlerhaft mit dem jeweiligen Maschinenergebnis $\widetilde{x} \in S(2, 53)$ berechnet wurden. Für alle $x \in [a, b]$ werden dabei die absoluten und relativen Fehlerschranken $\Delta(x)$ und $\varepsilon(x)$ als bekannt angesehen[5]:

$$|\widetilde{x} - x| \le \Delta(x), \qquad |\widetilde{x} - x| \le |x| \cdot \varepsilon(x), \quad \text{für alle } x \in [a, b] \in I\!I\!R$$

Setzt man $\Delta(x) = 0$ oder $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler ε_P ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a, b]$.

Bei vorausgesetzter hochgenauer Arithmetik liefert das Programm mit dem nachfolgenden Quelltext `Poly_relh.cpp` mit der `real`-Variablen `bnd` für das Polynom in (4.8) eine Oberschranke für den Auswertefehler

$$\left| \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)} \right| \le \varepsilon(P) = \texttt{bnd}, \quad u = (x - x_0)^k, \quad x \in [a, b] \in I\!I\!R; \tag{4.9}$$

[4] Diese Näherungen werden oft mit einem Computer-Algebrasystem berechnet.
[5] $\Delta(x)$ und $\varepsilon(x)$ sind Konstanten und keine von x abhängigen Größen!

```
// Programm:  Poly_relh.cpp Zur Abschätzung des relativen Polynom-
//            Auswertefehlers bei hochgenauer Arithmetik.
#include "bnd_util.hpp"   // Wegen Horn_relh(), Poly_Dat_rel(), ...
#include <iostream>       // Wegen cout
using namespace cxsc;
using namespace std;
int main()
{
// Polynomauswertung mit Hornerschema;  Gesucht: Rel. Auswertefehler;
// P(u) = a[0] + a[1]*u^1 + ... + a[N]*u^N,  u(x)=(x-x0)^k;
// a[j]: Exakte Koeffizienten, die durch Rundung von Dezimalwerten zur
//       jeweils nächsten Rasterzahl des IEEE-double-Formats entstehen
// Der Anwender muss die Werte von  N,k,x0  hier eingeben:
    int N=3, k=1;  // N : Polynomgrad; k: u(x)=(x-x0)^k, mit  1<=k<=6;
    real x0 = 3;   // x0: Entwicklungspunkt muss Maschinenzahl sein!!
    // Der Anwender muss die dezimalen Näherungen der Koeffizienten
    // hier eingeben: z.B. str0 = "108.1";  oder str0 = "1.01e-3";
    string str0 = "108";      string str1 = "54";
    string str2 = "12";       string str3 = "1";

    interval X;  // Intervall der exakten Polynom-Argumente x
    real bnd, diam, epsx;
    rvector a(0,N);  // a[j]: Exakte Koeffizienten aus S(2,53);

    str0 >> RndNext >> a[0];      str1 >> RndNext >> a[1];
    str2 >> RndNext >> a[2];      str3 >> RndNext >> a[3];
    // Die a[j] sind die zur nächsten Rasterzahl gerundeten strj
    while(1) // Endlosschleife
    {
        Poly_Dat_rel(X,epsx,diam);  // Eingabe von X,epsx,diam;
            // X   : Einschließung der exakten Polynom-Argumente
            // epsx: relative Fehlerschranke der gestörten Argumente
            // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
        bnd = Horn_relh(a,x0,X,epsx,k,diam);  // bnd: relative Fehler-
        // schranke bei nur hochgenauer Arithmetik beim Hornerschema
        cout << RndUp
             << "Relat. Fehlerschranke bei hochgenauer Arithmetik = "
             << bnd << endl << endl;
    }
}
```

Das umseitige Programm `Poly_relh` liefert mit `bnd` nach (4.9) eine Oberschranke des relativen Auswertefehlers für das Polynom

$$P_3(x-3) = 108 + 54 \cdot (x-3)^1 + 12 \cdot (x-3)^2 + (x-3)^3, \quad x \in [a,b]; \tag{4.10}$$

In die Quelldatei `Poly_relh.cpp` ist daher `int N=3, k=1;` und `real x0=3;` einzutragen. Außerdem müssen die dezimalen Näherungen der Polynomkoeffizienten eingetragen werden, die anschließend im Programm automatisch in die als rundungsfehlerfrei angenommenen Polynomkoeffizienten `a[j]` $\in S(2,53)$ gerundet werden. In der nicht abbrechende `while`-Schleife kann der Anwender das Intervall $[a,b]$ der exakten Polynomargumente x, die relative Fehlerschranke `epsx` $= \varepsilon(x)$ der gestörten Argumente $\widetilde{x}$ und den Parameter `diam` eingeben, mit dem die Intervallzerlegung zur Optimierung der relativen Fehlerschranke $\varepsilon(P)$ gesteuert wird. Mit `diam` $= 10^{-5}$ erhält man i.a. schon optimale Werte für $\varepsilon(P)$. Für jeden Parametersatz kann $\varepsilon(P)$ jeweils neu berechnet werden, ohne die Größen `N,k,x0` und die dezimalen Näherungen der Polynomkoeffizienten immer wieder eingeben zu müssen.

Wenn allerdings der relative Auswertefehler für ein anderes Polynom berechnet werden soll, so müssen neben `N,k,x0` auch die dezimalen Näherungen neu eingegeben werden, und das Programm ist vor dem ersten Aufruf natürlich neu zu übersetzen! Beachten Sie bitte, dass der dezimale Eingabewert für `x0` so gewählt werden muss, dass dieser rundungsfehlerfrei ins **IEEE**-double-Format gerundet werden kann!

Für das obige Polynom $P_3(x-3)$ erhält man mit $[a,b] = [1.5, 4.5]$, $\varepsilon(x) = 0$ und `diam` $= 10^{-6}$ die relative Fehlerschranke $\varepsilon(P) = 7.105435 \cdot 10^{-16}$.

4.3.1 Beispiele

In den folgenden Beispielen betrachten wir die in (4.8) definierten Polynome

$$\begin{aligned} P_N(u) &= a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + \ldots + a_N \cdot u^N, \\ u = u(x-x_0) &= (x-x_0)^k, \quad k \in \{1,2,3,4,5,6\}, \quad x \in [a,b] \in I\!I\!R; \end{aligned} \tag{4.11}$$

mit rundungsfehlerfreien, binären Koeffizienten a_j, die durch Rundung von dezimalen Näherungswerten zur jeweils nächsten Rasterzahl definiert sind. Mit dem Programm `Poly_relh` kann dann nach Eintragung von `N,k,x0` und der dezimalen Koeffizientennäherungen in `Poly_relh.cpp` eine Oberschranke $\varepsilon(P)$ des relativen Auswertefehlers

$$\left| \frac{P_N(u) - \widetilde{P}_N(\widetilde{u})}{P_N(u)} \right| \le \varepsilon(P) = \texttt{bnd}, \qquad u = x^k, \quad x \in [a,b] \in I\!I\!R \tag{4.12}$$

bei hochgenauer Arithmetik berechnet werden.

Beispiel 1 (Polynome mit binären Koeffizienten)
Wir betrachten die beiden identischen Polynome

$$P_3(x-3) := 108 + 54(x-3) + 12(x-3)^2 + (x-3)^3 \equiv Q_3(x) := 27 + 9x + 3x^2 + x^3,$$

die in der Umgebung von $x_0 = 3$ nach dem Hornerschema im **IEEE** double-Format auszuwerten sind. Es soll untersucht werden, wie sich der Polynomauswertefehler ändert, wenn man

- den Durchmesser des Argumentintervalls $|x-3| \leq \eta$ variiert
- an Stelle von $P_3(x-3)$ das ausmultiplizierte Polynom $Q_3(x)$ wählt.

Es wird angenommen, dass die Grundoperationen beim Hornerschema nur hochgenau ausgeführt werden. Hat man in `Poly_relh.cpp` mit einem Editor die jeweils richtigen Daten eingetragen, so erhält man nach der Übersetzung durch Aufruf des Programms `Poly_relh` für die relativen Auswertefehler $\varepsilon(Q)$ und $\varepsilon(P)$ folgende Oberschranken, die in Tabelle 4.4 zusammengestellt sind:

Relative Auswertefehler von $P_3(x-3) \equiv Q_3(x)$; $\varepsilon(x) = 0$			
$P_3(x-3)$		$Q_3(x) \equiv P_3(x-3)$	
$\lvert x-3\rvert \leq \eta$	$\varepsilon(P)$	$\lvert x-3\rvert \leq \eta$	$\varepsilon(Q)$
$x \in [1.5, 4.5]$	$7.105435 \cdot 10^{-16}$	$x \in [1.5, 4.5]$	$8.590486 \cdot 10^{-16}$
$x \in [2, 4]$	$5.909816 \cdot 10^{-16}$	$x \in [2, 4]$	$8.590486 \cdot 10^{-16}$
$x \in [2.5, 3.5]$	$3.650011 \cdot 10^{-16}$	$x \in [2.5, 3.5]$	$7.298582 \cdot 10^{-16}$
$x \in [2.9, 3.1]$	$2.453912 \cdot 10^{-16}$	$x \in [2.9, 3.1]$	$3.985568 \cdot 10^{-16}$
$x \in [2.99, 3.01]$	$2.242767 \cdot 10^{-16}$	$x \in [2.99, 3.01]$	$3.895946 \cdot 10^{-16}$

Tabelle 4.4: Relative Auswertefehler $\varepsilon(P), \varepsilon(Q)$, $\texttt{diam} = 10^{-6}$

Man erkennt, dass die Oberschranken $\varepsilon(P), \varepsilon(Q)$ der relativen Auswertefehler beider Polynome $P_3(x-3) \equiv Q_3(x)$ mit kleiner werdenden Intervalldurchmessern ebenfalls kleiner werden. Entwickelt man jedoch das Polynom $Q_3(x)$ an der Stelle $x_0 = 3$ in sein Taylorpolynom $P_3(x - x_0)$ und wertet man auf der Maschine dieses Polynom mit dem Argument $(x - x_0)$ aus, so erhält man deutlich kleinere Auswertefehler, insbesondere in der unmittelbaren Umgebung des Entwicklungspunktes $x_0 = 3$. Zur Minimierung des Polynomauswertefehlers ergibt sich damit die wichtige Grundregel:

Ist ein Polynom in der Umgebung von $x_0 \in S(2,53)$ auszuwerten, so reduziert sich der relative Auswertefehler ganz wesentlich, wenn dieses Polynom vorher im Punkt x_0 in ein Taylorpolynom entwickelt und das Argumentintervall mit dem Mittelpunkt x_0 nicht zu groß gewählt wird.

Beispiel 2 (Polynome mit binären Koeffizienten)
Wir betrachten zunächst das Polynom

$$D_4(u) := d_0 + d_1 \cdot u^1 + d_2 \cdot u^2 + d_3 \cdot u^3 + d_4 \cdot u^4, \quad u = x^2, \quad x \in [a,b] \in I\!I\!R$$

Die vorgegebenen dezimalen Koeffizienten

$$d_0 = 1.12837916709551256 \cdot 10^{+0} \qquad d1 = 1.35894887627277916 \cdot 10^{-1}$$
$$d_2 = 4.03259488531795274 \cdot 10^{-2} \qquad d3 = 1.20339380863079457 \cdot 10^{-3}$$
$$d_4 = 6.49254556481904354 \cdot 10^{-5}$$

sind zur jeweils nächsten Rasterzahl a_j des **IEEE** double-Formats zu runden. Das so entstandene Polynom

$$P_4(u) := a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + a_3 \cdot u^3 + a_4 \cdot u^4, \quad u = x^2, \quad x \in [a,b] \in I\!I\!R$$

ist in hochgenauer Arithmetik nach dem Hornerschema auszuwerten, und mit dem Programm `Poly_relh` ist für alle $x \in [a,b]$ eine Oberschranke $\varepsilon(P)$ des relativen Auswertefehlers zu berechnen. Wir nehmen dabei $\varepsilon(x) = 0$ an, so dass sich damit $\varepsilon(P)$ nur auf alle Maschinenzahlen des Argumentintervalls $[a,b]$ beziehen kann.

Vor der Übersetzung von `Poly_relh` müssen in `Poly_relh.cpp` mit einem Editor die Werte `int N=4, k=2; real x0=0;` und die obigen fünf dezimalen Näherungen d_j eingetragen werden. Nach dem Start des Programms und nach Eingabe der Werte $[a,b] = [10^{-10}, 0.65]$, $\varepsilon(x) = 0$, $\texttt{diam} = 10^{-6}$ erhält man mit

$$\left| \frac{P_3(u) - \widetilde{P}_3(\widetilde{u})}{P_3(u)} \right| \leq \varepsilon(P), \quad \text{für alle } x \in [10^{-10}, 0.65] \cap S(2,53);$$

für den relativen Auswertefehler die Oberschranke $\boxed{\varepsilon(P) = 2.622959 \cdot 10^{-16}}$

Beispiel 3 (Polynome mit binären Koeffizienten)
Wir betrachten zunächst das Polynom

$$D_4(u) := d_0 + d_1 \cdot u^1 + d_2 \cdot u^2 + d_3 \cdot u^3 + d_4 \cdot u^4, \quad u = x^2, \quad x \in [a,b] \in I\!I\!R$$

Die vorgegebenen dezimalen Koeffizienten

$$d_0 = 1.00000000000000000 \cdot 10^{+0} \qquad d_1 = 4.53767041780002545 \cdot 10^{-1}$$
$$d_2 = 8.69936222615385890 \cdot 10^{-2} \qquad d_3 = 8.49717371168693357 \cdot 10^{-3}$$
$$d_4 = 3.64915280629351082 \cdot 10^{-4}$$

sind zur jeweils nächsten Rasterzahl a_j des **IEEE** double-Formats zu runden. Das so entstandene Polynom

$$Q_4(u) := a_0 + a_1 \cdot u^1 + a_2 \cdot u^2 + a_3 \cdot u^3 + a_4 \cdot u^4, \quad u = x^2, \quad x \in [a,b] \in I\!I\!R$$

ist in hochgenauer Arithmetik nach dem Hornerschema auszuwerten, und mit dem Programm `Poly_relh` ist für alle $x \in [a,b]$ eine Oberschranke $\varepsilon(Q)$ des relativen Auswertefehlers zu berechnen. Wir nehmen dabei $\varepsilon(x) = 0$ an, so dass sich damit $\varepsilon(Q)$ nur auf alle Maschinenzahlen des Argumentintervalls $[a,b]$ beziehen kann.

Vor der Übersetzung von `Poly_relh` müssen in `Poly_relh.cpp` mit einem Editor die Werte `int N=4, k=2; real x0=0;` und die obigen fünf dezimalen Näherungen d_j eingetragen werden. Nach dem Start des Programms und nach Eingabe der Werte $[a,b] = [10^{-10}, 0.65]$, $\varepsilon(x) = 0$, $\texttt{diam} = 10^{-6}$ erhält man mit

$$\left| \frac{Q_3(u) - \widetilde{Q}_3(\widetilde{u})}{Q_3(u)} \right| \leq \varepsilon(Q), \quad \text{für alle } x \in [10^{-10}, 0.65] \cap S(2,53);$$

für den relativen Auswertefehler die Oberschranke $\boxed{\varepsilon(Q) = 3.077673 \cdot 10^{-16}}$

Anmerkung:

Mit den in den Beispielen 2 und 3 behandelten Polynomen $P_4(u)$ und $Q_4(u)$ kann die Fehlerfunktion $\texttt{erf}(x)$ für alle $x \in [10^{-10}, 0.65]$ durch eine rationale Funktion sehr gut approximiert werden:

$$\texttt{erf}() := \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \approx x \cdot \frac{P_4(x^2)}{Q_4(x^2)}, \quad x \in [10^{-10}, 0.65]$$

In Abschnitt 4.4 werden wir den Auswertefehler des obigen Polynomquotienten in einem Beispiel gesondert abschätzen.

Beispiel 4 (Polynome mit binären Koeffizienten)
Wir betrachten zunächst das Polynom

$$D_6(u) := d_0 + d_1 \cdot u^1 + \ \ldots \ + d_6 \cdot u^6, \quad u = x - 2, \quad x \in [1.5, 2.5]$$

Die vorgegebenen dezimalen Koeffizienten

$$d_0 = 1.00000000000000021 \cdot 10^{+0} \qquad d_1 = 5.69877694028250474 \cdot 10^{-1}$$
$$d_2 = 2.48388096138794478 \cdot 10^{-1} \qquad d_3 = 6.38386073132900548 \cdot 10^{-2}$$
$$d_4 = 1.41785707320153112 \cdot 10^{-2} \qquad d_5 = 1.85070207418211271 \cdot 10^{-3}$$
$$d_6 = 2.26359309735580607 \cdot 10^{-4}$$

sind zur jeweils nächsten Rasterzahl a_j des **IEEE** double-Formats zu runden. Das so entstandene Polynom

$$P_6(u) := a_0 + a_1 \cdot u^1 + \ \ldots \ + a_6 \cdot u^6, \quad u = x - 2, \quad x \in [1.5, 2.5]$$

ist in hochgenauer Arithmetik nach dem Hornerschema auszuwerten, und mit dem Programm `Poly_relh` ist für alle $x \in [1.5, 2.5]$ eine Oberschranke $\varepsilon(P)$ des relativen Auswertefehlers zu berechnen. Wir nehmen dabei $\varepsilon(x) = 0$ an, so dass sich damit $\varepsilon(P)$ nur auf alle Maschinenzahlen des Argumentintervalls $[1.5, 2.5]$ beziehen kann.

Vor der Übersetzung von `Poly_relh` müssen in `Poly_relh.cpp` mit einem Editor die Werte `int N=6, k=1; real x0=2;` und die obigen dezimalen Näherungen d_j eingetragen werden. Nach dem Start des Programms und nach Eingabe der Werte $[a, b] = [1.5, 2.5]$, $\varepsilon(x) = 0$, `diam` $= 10^{-6}$ erhält man mit

$$\left| \frac{P_6(u) - \widetilde{P}_6(\widetilde{u})}{P_6(u)} \right| \leq \varepsilon(P), \quad \text{für alle } x \in [1.5, 2.5] \cap S(2, 53);$$

für den relativen Auswertefehler die Oberschranke $\boxed{\varepsilon(P) = 3.910130 \cdot 10^{-16}}$

Beispiel 5 (Polynome mit binären Koeffizienten)
Wir betrachten zunächst das Polynom

$$D_6(u) := d_0 + d_1 \cdot u^1 + \ \ldots \ + d_6 \cdot u^6, \quad u = x - 2, \quad x \in [1.5, 2.5]$$

Die vorgegebenen dezimalen Koeffizienten

$$d_0 = 1.00000000000000000 \cdot 10^{+0} \qquad d_1 = 1.47093358929773866 \cdot 10^{-1}$$
$$d_2 = -2.25641002240105507 \cdot 10^{-1} \qquad d_3 = 1.70801916251380634 \cdot 10^{-2}$$
$$d_4 = 1.36369643727300423 \cdot 10^{-2} \qquad d_5 = -3.19656113959261051 \cdot 10^{-3}$$
$$d_6 = 2.07060796237616602 \cdot 10^{-4}$$

sind zur jeweils nächsten Rasterzahl a_j des **IEEE** double-Formats zu runden. Das so entstandene Polynom

$$Q_6(u) := a_0 + a_1 \cdot u^1 + \ \ldots \ + a_6 \cdot u^6, \quad u = x - 2, \quad x \in [1.5, 2.5]$$

ist in hochgenauer Arithmetik nach dem Hornerschema auszuwerten, und mit dem Programm `Poly_relh` ist für alle $x \in [1.5, 2.5]$ eine Oberschranke $\varepsilon(Q)$ des relativen Auswertefehlers zu berechnen. Wir nehmen dabei $\varepsilon(x) = 0$ an, so dass sich damit $\varepsilon(Q)$ nur auf alle Maschinenzahlen des Argumentintervalls $[1.5, 2.5]$ beziehen kann.

Vor der Übersetzung von `Poly_relh` müssen in `Poly_relh.cpp` mit einem Editor die Werte `int N=6, k=1; real x0=2;` und die obigen dezimalen Näherungen d_j eingetragen werden. Nach dem Start des Programms und nach Eingabe der Werte $[a, b] = [1.5, 2.5]$, $\varepsilon(x) = 0$, `diam` $= 10^{-6}$ erhält man mit

$$\left| \frac{Q_6(u) - \widetilde{Q}_6(\widetilde{u})}{Q_6(u)} \right| \leq \varepsilon(Q), \quad \text{für alle } x \in [1.5, 2.5] \cap S(2, 53);$$

für den relativen Auswertefehler die Oberschranke $\boxed{\varepsilon(Q) = 2.469759 \cdot 10^{-16}}$

Anmerkung:

Mit den in den Beispielen 4 und 5 behandelten Polynomen $P_6(u)$ und $Q_6(u)$ kann die Gammafunktion $\Gamma(x)$ für alle $x \in [1.5, 2.5]$ durch eine rationale Funktion sehr gut approximiert werden:

$$\Gamma(x) \approx \frac{P_6(x-2)}{Q_6(x-2)}, \quad x \in [1.5, 2.5]$$

Im Abschnitt 4.4 werden wir dann den relativen Auswertefehler der oben definierten rationalen Funktion im Beispiel 1 auf Seite 115 gesondert abschätzen.

4.4 Rationale Funktionen

Wir betrachten zunächst mit $u = (x - x_0)^k$ und $1 \leq k \leq 6$ die Polynome

$$\begin{aligned} Z_N(u) &= A_0 + A_1 \cdot u^1 + \ \ldots \ + A_N \cdot u^N \\ N_M(u) &= B_0 + B_1 \cdot u^1 + \ \ldots \ + B_M \cdot u^M \end{aligned}$$

mit gegebenen dezimalen Koeffizienten A_j, B_j, die z.B. mit Hilfe eines Computer-Algebrasystems bestimmt worden sind. Wir setzen voraus, dass die dezimalen Werte

A_j, B_j jeweils zur nächsten Rasterzahl des **IEEE** double-Systems gerundet werden. Dadurch definiert man die Polynome

$$\begin{aligned} P_N(u) &= a_0 + a_1 \cdot u^1 + \ \ldots \ + a_N \cdot u^N, \quad u = (x - x_0)^k \\ Q_M(u) &= b_0 + b_1 \cdot u^1 + \ \ldots \ + b_M \cdot u^M, \quad 1 \le k \le 6 \end{aligned}$$

mit binären Koeffizienten $a_j.b_j$, die als rundungsfehlerfrei anzusehen sind. Mit diesen Polynomen wird die folgende rationale Funktion $R(u)$ definiert:

$$R(u) := \frac{P_N(u)}{Q_M(u)}, \quad u = (x - x_0)^k, \quad 1 \le k \le 6, \quad x \in [a, b] \in I\!I\!R; \tag{4.13}$$

Vorausgesetzt wird zusätzlich: $x_0 \in S(2, 53)$ und $Q_M(u) \neq 0$ für alle $x \in [a, b]$. Bezeichnet man mit $\widetilde{u}$ die auf der Maschine ausgewertete Potenz $(\widetilde{x} - x_0)^k$, so gilt: $\widetilde{u} = (\widetilde{x} \ominus x_0) \odot (\widetilde{x} \ominus x_0) \odot \ldots$, wobei die von k abhängige Auswertevorschrift auf Seite 93 angegeben ist. Bedeutet dann $\widetilde{R}(\widetilde{u})$ den für das Maschinenargument $\widetilde{x}$ auf der Maschine ausgewerteten Funktionswert, so gilt für den relativen Auswertefehler:

$$\varepsilon_R := \frac{R(u) - \widetilde{R}(\widetilde{u})}{R(u)}, \qquad u = (x - x_0)^k, \quad x \in [a, b] \in I\!I\!R;$$

Wir interpretieren dabei $[a, b]$ als den exakten Wertebereich einer Funktion mit den exakten Funktionswerten x und nehmen weiter an, dass diese x auf dem Computer fehlerhaft mit dem jeweiligen Maschinenergebnis $\widetilde{x} \in S(2, 53)$ berechnet wurden. Für alle $x \in [a, b]$ werden dabei die relativen Fehlerschranken $\varepsilon(x)$ als bekannt angesehen[6]:

$$|\widetilde{x} - x| \le |x| \cdot \varepsilon(x), \quad \text{für alle } x \in [a, b] \in I\!I\!R$$

Setzt man $\varepsilon(x) = 0$ voraus, so bedeutet dies $x \equiv \widetilde{x}$, d.h. der relative Auswertefehler ε_R ist dann nur definiert für alle Maschinenzahlen $\widetilde{x} \in [a, b]$.

Bei vorausgesetzter hochgenauer Arithmetik liefert das Programm mit dem nachfolgenden Quelltext `Rat_relh.cpp` mit der `real`-Variablen `bnd` für die rationale Funktion in (4.13) eine Oberschranke für den Auswertefehler

$$\left| \frac{R(u) - \widetilde{R}(\widetilde{u})}{R(u)} \right| \le \varepsilon(R) = \texttt{bnd}, \quad u = (x - x_0)^k, \quad x \in [a, b] \in I\!I\!R; \tag{4.14}$$

Vor der Übersetzung des Programms muss der Anwender mit einem Editor in den Quelltext `Rat_relh.cpp` die entsprechenden Werte M, N, k, x_0 und die dezimalen Koeffizienten A_j, B_j eingeben. Die Rundung zu den jeweils nächsten Rasterzahlen $a_j, b_j \in S(2, 53)$ erfolgt dann im Programm `Rat_relh` automatisch.

[6] $\varepsilon(x)$ ist eine Konstanten und keine von x abhängigen Größe!

```
// Programm: Rat_relh.cpp Zur Abschätzung des Auswertefehlers einer
//           rationalen Funktion bei hochgenauer Arithmetik.
#include "bnd_util.hpp"   // Wegen Rat_relh(), Poly_Dat_rel(), ...
#include <iostream>       // Wegen cout
using namespace cxsc;   using namespace std;
int main()
{
// Gesucht: Relativer Auswertefehler einer rationalen Funktion
//          R(u) = P_N(u)/Q_M(u),  u = (x-x0)^k,  1<=k<=6;;
// P_N(u) = a[0] + a[1]*u^1 + ... + a[N]*u^N,  u = (x-x0)^k;
// Q_M(u) = b[0] + b[1]*u^1 + ... + b[M]*u^M,  u = (x-x0)^k;
// a[j],b[j]: Exakte Koeffizienten, die durch Rundung von
//            Dezimalwerten zur jeweils nächsten Rasterzahl des
//            IEEE-double-Formats entstehen.
// Der Anwender muss die Werte von  N,M,k,x0  hier eingeben:
    int N=6, M=6, k=1;  // N,M : Polynomgrad;
                        // u(x)=(x-x0)^k, mit 1<=k<=6;
    real x0 = 2;   // x0: Entwicklungspunkt muss Maschinenzahl sein!
    // Der Anwender muss die dezimalen Näherungen der Koeffizienten
    // von P_N(u) hier eingeben:  z.B. strP0 = "1.01e-3";
    string strP0 = "1.00000000000000021e+0";
    string strP1 = "5.69877694028250474e-1";
    string strP2 = "2.48388096138794478e-1";
    string strP3 = "6.38386073132900548e-2";
    string strP4 = "1.41785707320153112e-2";
    string strP5 = "1.85070207418211271e-3";
    string strP6 = "2.26359309735580607e-4";

    // Eingabe der dezimal. Näherungen der Koeffizienten von Q_M(u):
    string strQ0 = "1.00000000000000000";
    string strQ1 = "1.47093358929773866e-1";
    string strQ2 = "-2.25641002240105507e-1";
    string strQ3 = "1.70801916251380634e-2";
    string strQ4 = "1.36369643727300423e-2";
    string strQ5 = "-3.19656113959261051e-3";
    string strQ6 = "2.07060796237616602e-4";
    interval X;  // Intervall der exakten Polynom-Argumente x
    real bnd, diam, epsx;
    rvector a(0,N),b(0,M); // a[j],b[j]: Exakte Koeffztn. aus S(2,53);
```

```
    strP0 >> RndNext >> a[0];       strQ0 >> RndNext >> b[0];
    strP1 >> RndNext >> a[1];       strQ1 >> RndNext >> b[1];
    strP2 >> RndNext >> a[2];       strQ2 >> RndNext >> b[2];
    strP3 >> RndNext >> a[3];       strQ3 >> RndNext >> b[3];
    strP4 >> RndNext >> a[4];       strQ4 >> RndNext >> b[4];
    strP5 >> RndNext >> a[5];       strQ5 >> RndNext >> b[5];
    strP6 >> RndNext >> a[6];       strQ6 >> RndNext >> b[6];
    // Die a[j],b[j] sind die zur jeweils nächsten Rasterzahl
    // gerundeten Strings strPj,strQj,  j=0,1,...,6;
    while(1) // Endlosschleife
    {
        Poly_Dat_rel(X,epsx,diam);  // Eingabe von X,epsx,diam;
            // X   : Einschließung der exakten Polynom-Argumente
            // epsx: relative Fehlerschranke der gestörten Argumente
            // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
        bnd = Rat_relh(a,b,x0,X,epsx,k,diam); // bnd: relative Fehler-
        // schranke bei nur hochgenauer Arithmetik beim Hornerschema
        cout << "Relative Fehlerschranke bei hochgenauer Arithmetik ="
             << RndUp << bnd << endl << endl;
    }
}
```

4.4.1 Beispiele

Beispiel 1 (Rationale Funktionen mit binären Koeffizienten)
Wir betrachten die beiden Polynome

$$\begin{aligned} P_6(x-2) &= a_0 + a_1 \cdot (x-2)^1 + \ \ldots \ + a_6 \cdot (x-2)^6 \\ Q_6(x-2) &= b_0 + b_1 \cdot (x-2)^1 + \ \ldots \ + b_6 \cdot (x-2)^6 \end{aligned}$$

n	$a_n :=$ nearest(.)	$b_n :=$ nearest(.)
0	$+1.00000000000000021 \cdot 10^{+0}$	$+1.00000000000000000 \cdot 10^{+0}$
1	$+5.69877694028250474 \cdot 10^{-1}$	$+1.47093358929773866 \cdot 10^{-1}$
2	$+2.48388096138794478 \cdot 10^{-1}$	$-2.25641002240105507 \cdot 10^{-1}$
3	$+6.38386073132900548 \cdot 10^{-2}$	$+1.70801916251380634 \cdot 10^{-2}$
4	$+1.41785707320153112 \cdot 10^{-2}$	$+1.36369643727300423 \cdot 10^{-2}$
5	$+1.85070207418211271 \cdot 10^{-3}$	$-3.19656113959261051 \cdot 10^{-3}$
6	$+2.26359309735580607 \cdot 10^{-4}$	$+2.07060796237616602 \cdot 10^{-4}$

In der vorausgehenden Tabelle sind wie im Programmquellcode `Rat_relh.cpp` die gleichen Dezimalwerte eingetragen, die nach dem Start des Programms `Rat_relh` zur jeweils nächsten Rasterzahl a_j = `a[j]` bzw. b_j = `b[j]` gerundet werden. Im gleichen Quellcode findet man außerdem die Eintragungen: `N=6, M=6, k=1, x0=2`. Für die rationale Funktion

$$R(x-2) = \frac{P_6(x-2)}{Q_6(x-2)}$$

erhält man mit dem Programm `Rat_relh` nach Eingabe von $[a,b] = [1.5, 2.5]$, $\varepsilon(x) = 0$ und `diam` $= 10^{-6}$ für den relativen Auswertefehler die Oberschranke:

$$\varepsilon(R) = 8.341628 \cdot 10^{-16}, \qquad x = \widetilde{x} \in [1.5, 2.5] \cap S(2, 53)$$

Beachten Sie bitte, dass sich die Fehlerschranke $\varepsilon(R)$ wegen $\varepsilon(x) = 0$ nur auf die Rasterzahlen $x = \widetilde{x} \in [1.5, 2.5] \cap S(2, 53)$ des Argumentintervalls beziehen kann und dass die Polynome P_6, Q_6 in nur hochgenauer Arithmetik nach dem Hornerschema auszuwerten sind.

Beispiel 2 (Rationale Funktionen mit binären Koeffizienten)
Wir betrachten die beiden Polynome

$$\begin{aligned} P_4(u) &= a_0 + a_1 \cdot u^1 + \ \ldots\ + a_4 \cdot u^4, \quad u = x^2 \\ Q_4(u) &= b_0 + b_1 \cdot u^1 + \ \ldots\ + b_4 \cdot u^6 \end{aligned}$$

n	$a_n :=$ nearest(.)	$b_n :=$ nearest(.)
0	$1.12837916709551256 \cdot 10^{+0}$	$1.00000000000000000 \cdot 10^{+0}$
1	$1.35894887627277916 \cdot 10^{-1}$	$4.53767041780002545 \cdot 10^{-1}$
2	$4.03259488531795274 \cdot 10^{-2}$	$8.69936222615385890 \cdot 10^{-2}$
3	$1.20339380863079457 \cdot 10^{-3}$	$8.49717371168693357 \cdot 10^{-3}$
4	$6.49254556481904354 \cdot 10^{-5}$	$3.64915280629351082 \cdot 10^{-4}$

und definieren damit die rationale Funktion

$$R(u) = \frac{P_4(u)}{Q_4(u)}, \quad u = x^2, \quad x \in [a,b] = [10^{-10}, 0.65]$$

Um mit $\varepsilon(x) = 0$ für $x \in [a,b] = [10^{-10}, 0.65]$ eine Oberschranke $\varepsilon(R)$ des relativen Auswertefehlers zu berechnen, sind im Quelltext `Rat_relh.cpp` neben den Werten `N=4, M=4, k=2, x0=0` auch die dezimalen Näherungen der Polynomkoeffizienten aus obiger Tabelle entsprechend einzutragen. Nach dem Programmstart werden diese Näherungen automatisch in die jeweils nächsten Rasterzahlen a_j, b_j gerundet, die

dann als rundungsfehlerfreie Polynomkoeffizienten betrachtet werden. Nach Eingabe von $[a,b] = [10^{-10}, 0.65]$, $\varepsilon(x) = 0$, `diam` $= 10^{-5}$ erhält man als Schranke des relativen Auswertefehlers:

$$\varepsilon(R) = 7.921098 \cdot 10^{-16}, \qquad x = \widetilde{x} \in [10^{-10}, 0.65] \cap S(2,53)$$

Beachten Sie bitte, dass sich die obige Fehlerschranke $\varepsilon(R)$ wegen $\varepsilon(x) = 0$ nur auf die Rasterzahlen $x = \widetilde{x} \in [10^{-10}, 0.65] \cap S(2,53)$ des Argumentintervalls beziehen kann und dass die Polynome $P_4(u), Q_4(u)$ in nur hochgenauer Arithmetik nach dem Hornerschema auszuwerten sind.

Beispiel 3 (Rationale Funktionen mit binären Koeffizienten)
Wir betrachten wie im Beispiel 2 die gleichen Polynome

$$\begin{aligned} P_4(u) &= a_0 + a_1 \cdot u^1 + \ \ldots\ + a_4 \cdot u^4, \quad u = x^2 \\ Q_4(u) &= b_0 + b_1 \cdot u^1 + \ \ldots\ + b_4 \cdot u^6 \end{aligned}$$

n	$a_n :=$ nearest(.)	$b_n :=$ nearest(.)
0	$1.12837916709551256 \cdot 10^{+0}$	$1.00000000000000000 \cdot 10^{+0}$
1	$1.35894887627277916 \cdot 10^{-1}$	$4.53767041780002545 \cdot 10^{-1}$
2	$4.03259488531795274 \cdot 10^{-2}$	$8.69936222615385890 \cdot 10^{-2}$
3	$1.20339380863079457 \cdot 10^{-3}$	$8.49717371168693357 \cdot 10^{-3}$
4	$6.49254556481904354 \cdot 10^{-5}$	$3.64915280629351082 \cdot 10^{-4}$

und definieren damit die rationale Funktion

$$R(u) = x \cdot \frac{P_4(u)}{Q_4(u)}, \quad u = x^2, \quad x \in [a,b] = [10^{-10}, 0.65],$$

wobei der Quotient $P_4(u)/Q_4(u)$ jetzt noch mit dem Faktor x zu multiplizieren ist. Im folgenden Quelltext des **C-XSC** Programms `Rat_F.cpp` wird die Abschätzung des relativen Auswertefehlers für ein Teilintervall `Xi` mit der Funktion `Rat_F_Xi` vorgenommen. Um mit $\varepsilon(x) = 0$ für $x \in [a,b] = [10^{-10}, 0.65]$ eine Oberschranke $\varepsilon(R)$ des relativen Auswertefehlers zu berechnen, sind im Quelltext `Rat_F.cpp` neben den Werten `N=4, M=4, k=2, x0=0` auch die dezimalen Näherungen der Polynomkoeffizienten aus der letzten Tabelle entsprechend einzutragen. Nach dem Programmstart werden diese Näherungen automatisch in die jeweils nächsten Rasterzahlen a_j, b_j gerundet, die dann als rundungsfehlerfreie Polynomkoeffizienten betrachtet werden.

```
// Programm:   Rat_F.cpp zur Abschätzung des Auswertefehlers einer
// rationalen Funktion mit einem Faktor bei hochgenauer Arithmetik.
#include "bnd_util.hpp"   // Wegen Rat_F(), Poly_Dat_rel(), ...
#include <iostream>       // Wegen cout

using namespace cxsc;
using namespace std;

real Rat_F_Xi(const interval& Xi, const real& epsx,
              const ivector& ia, const ivector& ib,
              const int& k, const real& x0)
// x * P_N(u)/Q_M(u);  u = (x-x0)^k,  1<=k<=6;
// Berechnung der relat. Fehlerschranke über einem Teilintervall Xi,
// welches die exakten Polynom-Argumente enthält.
// Vorausgesetzt wird eine nur hochgenauer Arithmetik,
// mit u(x) = (x-x0)^k.
// epsx ist die relat. Fehlerschranke der gestörten Polynomargumente.
// ia,ib sind Punktintervalle und importieren die optimalen Einschlie-
// ßungen der exakten Polynomkoeffizienten von P_N(u) bzw. Q_M(u).

{
    real del1,del2,delu,delK;
    interval Kl,u,h1,h2;
    abs_subh3(Xi,epsx,interval(x0),0,Kl,delK);
    // Kl  : Einschließung von (x-x0) für alle x aus Xi;
    // delK: Absolute Fehlerschranke bei Auswertung von x-x0, wobei
    //       die relative Fehlerschranke epsx der gestörten
    //       Polynomargumente berücksichtigt wird.
    // Auswertung von u(x) = (x-x0)^k:
    switch (k)
    {
        case 1:  // u(x) = (x-x0)^1
            u = Kl;  delu = delK;
            break;
        case 2:  // u(x) = (x-x0)^2
            abs_mulh1(Kl,delK,Kl,delK,u,delu);
            break;
        case 3:
            abs_mulh1(Kl,delK,Kl,delK,h1,del1);
```

```
                            // h1: Einschließg. für Kl*Kl
        abs_mulh1(h1,del1,Kl,delK,u,delu);
                            // u: Einschg. für (Kl*Kl)*Kl
        break;
    case 4:  // u(x) = x^4
        abs_mulh1(Kl,delK,Kl,delK,h1,del1);
                            // h1: Einschließg. für Kl*Kl
        abs_mulh1(h1,del1,h1,del1,u,delu);
                            // u:  Einschlg. für (Kl*Kl)^2
        break;
    case 5:  // u(x) = x^5
        abs_mulh1(Kl,delK,Kl,delK,h1,del1);
                            // h1: Einschließg. für Kl*Kl
        abs_mulh1(h1,del1,h1,del1,h2,del2);
                            // h2: Einschlg. für (Kl*Kl)^2
        abs_mulh1(h2,del2,Kl,delK,u,delu);
                            // u:  Einschlg. für Kl^5
        break;
    case 6:  // u(x) = x^6
        abs_mulh1(Kl,delK,Kl,delK,h1,del1);  // Kl*Kl in h1
        abs_mulh1(h1,del1,h1,del1,h2,del2);  // (Kl*Kl)^2 in h2
         abs_mulh1(h2,del2,h1,del1,u,delu);
                            // u:  Einschlg. für Kl^6
        break;
    default:
        cerr << "Horn_relh_Xi(): Unzulässige Potenz x^k" << endl;
        exit(1);
} // switch
// u ist Einschließung von (Xi-x0)^k;
// delu ist die zugehörige absolute Fehlerschranke;
h1 = ia[Ub(ia)]; // h1: Horner-Startwert für das Zählerpolynom
del1 = 0;  // Die Polynomkoeffizienten sind rundungsfehlerfrei!
int m = Ub(ia)-1;
for (int j=m; j>=0; j--)
{
    abs_mulh1(h1,del1,u,delu,h2,del2);          // h2 = y*u
    if (ia[j]==1) abs_1px_delh(h2,del2,h1,del1);
    else abs_addh1(ia[j],0,h2,del2,h1,del1); // h1 = y*u + a[j]
}
```

```
//*****************************************************************
// -------- Berücksichtigung des Faktors x im Zähler durch: ----------
    abs_mulh3(Xi,epsx,h1,del1,Kl,delK);
//     Kl  : Einschließung von Xi*Zählerpolynomwert
//     delK: Absolute Fehlerschranke des Zählers  Xi*P_N(u)
//*****************************************************************
    h1 = ib[Ub(ib)]; // h1: Horner-Startwert für Nennerpolynoms
    del1 = 0;  // Die Polynomkoeffizienten sind rundungsfehlerfrei!
    m = Ub(ib)-1;
    for (int j=m; j>=0; j--)
    {
        abs_mulh1(h1,del1,u,delu,h2,del2);          // h2 = y*u
        if (ib[j]==1) abs_1px_delh(h2,del2,h1,del1);
        else abs_addh1(ib[j],0,h2,del2,h1,del1); // h1 = y*u + a[j]
    } // del1: absoluter Fehler;  h1: Einschließung des Nennerpolynoms

    // Berechnung des relativen Fehlers des Polynom-Quotienten:
    rel_divh1(Kl,delK,h1,del1,h2,del2);
    return del2;  // Rückgabe der relativen Fehlerschranke del2
} // Rat_F_Xi
real Rat_F(const rvector& a, const rvector& b, const real& x0,
           const interval& X, const real& epsx, const int& k,
           const real& diam)
// Liefert mit bnd eine Oberschranke des relativen Fehlers, wenn der
// Quotient    P_N(u)/Q_M(u),  u=(x-x0)^k,  1<=k<=6, mit den
// Polynomen:  P_N(u) = a[0] + a[1]*u^1 +...+a[N]*u^N,
//             Q_M(u) = b[0] + b[1]*u^1 +...+b[M]*u^M,
// zusätzlich noch mit einem Faktor, z.B. x, multipliziert und die
// Polynome nach dem Hornerschema ausgewertet werden.
// Wie dieser Faktor bei der Fehlerabschätzung zu berücksichtigen
// ist, zeigt die Implementierung der Funktion Rat_F_Xi() im Quelltext
// des Programms Rat_F.cpp
// a ist der Vektor der exakten Koeffizienten a[j], j=0,1,...,N;
// b ist der Vektor der exakten Koeffizienten b[j], j=0,1,...,M;
// x0 ist der Entwicklungspunkt; x0 in S(2,53) muss erfüllt sein!
// X ist das Intervall der exakten Polynomargumente x aus X. Wurden
// die Polynomargumente fehlerhaft berechnet, so ist epsx die
// entsprechende relative Fehlerschranke.
// X wird in Teilintervalle Xi zerlegt, um bnd zu minimieren; diam
// bestimmt den Durchmesser der X1.      diam = 10^(-5) ist i.a. OK.
```

```
{
    real bnd;
    ivector ia(0,Ub(a)), ib(0,Ub(b));  // Einschließungen der exakten
                      // Polynom-Koeffizienten durch Punktintervalle
    for (int j=0; j<=Ub(a); j++) ia[j] = interval(a[j]);
    for (int j=0; j<=Ub(b); j++) ib[j] = interval(b[j]);
    Max_Rat_Xi(Rat_F_Xi,X,epsx,diam,ia,ib,k,x0,bnd);
    return bnd;
} // Rat_F

int main()
{
// Gesucht: Relativer Auswertefehler einer Funktion
//          R(u) = x*P_N(u)/Q_M(u),  u = (x-x0)^k,  1<=k<=6;
// P_N(u) = a[0] + a[1]*u^1 + ... + a[N]*u^N,  u = (x-x0)^k;
// Q_M(u) = b[0] + b[1]*u^1 + ... + b[M]*u^M,  u = (x-x0)^k;
// a[j],b[j]: Exakte Koeffizienten, die durch Rundung von
//            Dezimalwerten zur jeweils nächsten Rasterzahl des
//            IEEE-double-Formats entstehen.
// Der Anwender muss die Werte von  N,M,k,x0  hier eingeben:
    int N=4, M=4, k=2;  // N,M : Polynomgrad;
                        // u(x)=(x-x0)^k, mit 1<=k<=6;
    real x0 = 0;   // x0: Entwicklungspunkt muss Maschinenzahl sein!
    // Der Anwender muss die dezimalen Näherungen der Koeffizienten
    // von P_N(u) hier eingeben:    z.B. strP0 = "1.01e-3";
    string strP0 = "1.12837916709551256e+0";
    string strP1 = "1.35894887627277916e-1";
    string strP2 = "4.03259488531795274e-2";
    string strP3 = "1.20339380863079457e-3";
    string strP4 = "6.49254556481904354e-5";

    // Der Anwender muss die dezimalen Näherungen der Koeffizienten
    // von Q_M(u) hier eingeben:  z.B. strQ0 = "-1.22301";
    string strQ0 = "1.00000000000000000";
    string strQ1 = "4.53767041780002545e-1";
    string strQ2 = "8.69936222615385890e-2";
    string strQ3 = "8.49717371168693357e-3";
    string strQ4 = "3.64915280629351082e-4";
    interval X;  // Intervall der exakten Polynom-Argumente x
```

```
    real bnd, diam, epsx;
    rvector a(0,N),b(0,M); // a[j],b[j]: Exakte Koeffztn. aus S(2,53);

    strP0 >> RndNext >> a[0];
    strP1 >> RndNext >> a[1];
    strP2 >> RndNext >> a[2];
    strP3 >> RndNext >> a[3];
    strP4 >> RndNext >> a[4];

    strQ0 >> RndNext >> b[0];
    strQ1 >> RndNext >> b[1];
    strQ2 >> RndNext >> b[2];
    strQ3 >> RndNext >> b[3];
    strQ4 >> RndNext >> b[4];
    // Die a[j],b[j] sind die zur jeweils nächsten Rasterzahl
    // gerundeten Strings strPj,strQj,  j = 0,1,...,4;

    while(1) // Endlosschleife
    {
        Poly_Dat_rel(X,epsx,diam);  // Eingabe von X,epsx,diam;
            // X   : Einschließung der exakten Polynom-Argumente
            // epsx: relative Fehlerschranke der gestörten Argumente
            // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
        bnd = Rat_F(a,b,x0,X,epsx,k,diam); // bnd: relative Fehler-
        // schranke bei nur hochgenauer Arithmetik beim Hornerschema
        cout << RndUp
             << "Relat. Fehlerschranke bei hochgenauer Arithmetik = "
             << bnd << endl << endl;
    }
}
```

Nach Eingabe von $[a,b] = [10^{-10}, 0.65]$, $\varepsilon(x) = 0$, `diam` $= 10^{-5}$ erhält man als Schranke des relativen Auswertefehlers:

$$\varepsilon(R) = 1.014163 \cdot 10^{-15}, \qquad x = \widetilde{x} \in [10^{-10}, 0.65] \cap S(2,53)$$

Beachten Sie bitte, dass sich die obige Fehlerschranke $\varepsilon(R)$ wegen $\varepsilon(x) = 0$ nur auf die Rasterzahlen $x = \widetilde{x} \in [10^{-10}, 0.65] \cap S(2,53)$ des Argumentintervalls beziehen kann und dass die Auswertung der Polynome $P_4(u), Q_4(u)$ nach dem Hornerschema und die Multiplikation mit dem Faktor x nach Voraussetzung in nur hochgenauer Arithmetik erfolgt.

Anmerkungen:

- Falls Sie mit diesem Programm eine Schranke des relativen Auswertefehlers für eine andere rationale Funktion mit einem anderen Faktor berechnen wollen, so sollten Sie vorher eine Sicherheitskopie dieses Programms `Rat_F.cpp` anlegen!
- Im obigen Quelltext der Funktion `Rat_F_Xi()` wird der Faktor x durch den Aufruf der Funktion `abs_mulh3(Xi,epsx,h1,del1,Kl,delK)` berücksichtigt. Zur Bedeutung der letzten Namensziffer `3` vergleichen Sie bitte die dritte Zeile in Tabelle 2.1 auf Seite 24, nach der `epsx` eine relative und `del1` eine absolute Fehlerschranke sein muss und in diesem Fall auch realisiert wird.
- Die berechnete Fehlerschranke $\varepsilon(R) = 1.014163 \cdot 10^{-15}$ erhält man auch dann, wenn man die Fehlerschranken von $P_4(u)/Q_4(u)$ und bez. der Multiplikation mit x einzeln berechnet und dann zu einer einzigen Fehlerschranke zusammenfasst. Nach der Anmerkung von Seite 94 sollte man jedoch die Fehlerabschätzung grundsätzlich für den vollständigen Term durchführen!

Abschließend noch eine allgemeine Anmerkung zur Fehlerabschätzung: In den letzten Beispielen haben wir Programme vorgestellt, mit denen man relative Auswertefehler von Polynomen oder rationalen Funktionen abschätzen konnte. Die berechneten Fehlerschranken sind zunächst einmal stets **garantierte** Oberschranken bei einer angenommenen hochgenauen Arithmetik der Grundoperationen. In Spezialfällen kann man diese Fehlerschranken jedoch noch weiter verkleinern, wenn man den Algorithmus zur Auswertung eines Hornerschritts gezielt abändert.

So könnte man die in einem Hornerschritt auftretende Multiplikation und die anschließende Addition des entsprechenden Polynomkoeffizienten im Akkumulator exakt auswerten. Rundet man dann den Akku-Inhalt zur nächsten Rasterzahl des **IEEE** double-Formats, so liegt der Hornerschritt mit einer relativen Genauigkeit von nur $\varepsilon(m) = \frac{1}{2} \cdot 2^{-52}$ vor. Da die Akkuoperationen jedoch softwaremäßig, d.h. sehr zeitaufwendig simuliert werden müssen, wird die kleinere Fehlerschranke mit dem Einsatz des Akkumulators viel zu teuer erkauft.

Viel effektiver kann man bei der Polynomauswertung jedoch vorgehen, wenn z.B. das Polynomargument $\mathtt{x}$ zu quadrieren und der Term $\frac{1}{2} \cdot \mathtt{x}^2$ auszuwerten ist. Zerlegt man dann $\mathtt{x} = \mathtt{u} + \mathtt{v}$ so in zwei Maschinenzahlen $\mathtt{u}, \mathtt{v} \in S(2, 53)$, dass $\mathtt{u} = \mathtt{x}|_{26}$ gilt[7], so erhält man mit $\mathtt{v} := \mathtt{x} \ominus \mathtt{u}$:

$$\mathtt{x} = \mathtt{u} + \mathtt{v} = \mathtt{u} \oplus \mathtt{v}, \quad \mathtt{v} := \mathtt{x} \ominus \mathtt{u} = \mathtt{x} - \mathtt{u} \quad \text{und} \quad \mathtt{u} \odot \mathtt{u} = \mathtt{u} \cdot \mathtt{u}$$

Im nächsten Abschnitt werden wir zeigen, dass man mit dieser Zerlegung die Fehlerschranken deutlich reduzieren kann.

[7] $\mathtt{x}|_{26}$ stimmt mit $\mathtt{x}$ auf den ersten 26 Mantissenbits überein, die restlichen 53-26=27 Bits werden mit Hilfe der Funktion `Cut26()` auf Null gesetzt.

4.5 Spezielle Terme

In diesem Abschnitt stellen wir anhand einiger Beispiele alle Programmwerkzeuge vor, um zu einem vorgegebenen Term $T(x)$ eine garantierte Schranke des relativen Auswertefehlers zu berechnen. Dabei wird vorausgesetzt, dass die verwendeten Grundoperationen alle in nur hochgenauer Arithmetik durchgeführt werden und dass sich die berechneten Fehlerschranken nur auf die Maschinenzahlen x eines vorgegebenen Argumentintervalls beziehen sollen. Vorausgesetzt wird also $\varepsilon(x) = \Delta(x) = 0$, d.h. $x \in [a,b] \cap S(2,53)$. In diesem Abschnitt sind also u, v, x, y, z Maschinenzahlen des **IEEE** double-Formats, und mit $\widetilde{z}$ bezeichnen wir ebenfalls eine Maschinenzahl, die jedoch mit Hilfe der Grundoperationen in nur hochgenauer Arithmetik fehlerhaft berechnet wurde.

4.5.1 $x + \frac{1}{2}\,x^2$

Der Term $T(x) = x + \frac{1}{2} \cdot x^2$ ist für alle Maschinenzahlen $x \in [a,b] = [0.125, 2]$ auszuwerten, und für den dabei auftretenden relativen Auswertefehler ist eine garantierte Oberschranke $\varepsilon(T)$ zu berechnen:

$$\left| \frac{T(x) - \widetilde{T}(x)}{T(x)} \right| \leq \varepsilon(T), \quad x \in [a,b] = [0.125, 2] \tag{4.15}$$

$\widetilde{T}(x)$ ist dabei das i.a. fehlerbehaftete Maschinenergebnis, wenn $T(x)$ in hochgenauer Arithmetik ausgewertet wird. Da die Auswertung des Terms $T(x)$ nicht eindeutig ist, werden wir in den Unterabschnitten 4.5.1.1 bis 4.5.1.4 jeweils einen der vier folgenden Algorithmen untersuchen und jeweils den entsprechenden Auswertefehler abschätzen:

1. $T_1(x) := x \cdot (1 + \frac{1}{2}\,x)$
2. $T_2(x) := x + \frac{1}{2}\,x^2$
3. $T_{3,4}(x) := x + \frac{1}{2}\,x^2,\ x = u + v$, wobei die Maschinenzahl $u := x|_{24}$ mit x auf den ersten 24 Mantissen-Bits[8] übereinstimmt und die restlichen $53 - 24 = 29$ Bits verschwinden. $T_3(x)$ und $T_4(x)$ unterscheiden sich durch die verschiedenen Möglichkeiten, die auftretenden Summanden zusammenzufassen.

Die obigen Algorithmen sind dabei so ausgewählt, dass $T_1(x)$ den größten und $T_4(x)$ den kleinsten relativen Auswertefehler liefert. Leider wird bei $T_4(x)$ dieser kleinere

[8] Bei diesen 24 Mantissen-Bits ist das so genannte hidden bit mitgezählt, so dass die ersten 23 real gespeicherten Mantissen-Bits mit denen von x übereinstimmen. $u = x|_{24}$ wird mit dem Funktionsaufruf `u=Cut24(x)` berechnet.

Auswertefehler durch eine etwas größere Laufzeit erkauft, da die Berechnung von $\frac{1}{2}\,x^2$ durch die Zerlegung von $x = u + v$ in doppelter Genauigkeit erfolgt.

4.5.1.1 $T_1(x) = x \cdot (1 + \frac{1}{2}\, x)$

Für den ersten Term $T_1(x) := x \cdot (1 + \frac{1}{2}\, x)$ ist nach (4.15) für alle Maschinenzahlen $x \in [0.125, 2]$ eine Oberschranke $\varepsilon(T_1)$ des relativen Auswertefehlers zu berechnen. Im nachfolgenden Programm `Term_relh` ist dazu lediglich in der Funktion `Term()` die Berechnung der Fehlerschranke für ein Teilintervall `Xi` zu implementieren.

```
// Programm:  Term_relh.cpp zur Abschätzung des relativen
//            Auswertefehlers bei hochgenauer Arithmetik.
#include "bnd_util.hpp"   // Wegen Term_relh(), Term_Dat(), ...
#include <iostream>       // Wegen cout
using namespace cxsc;
using namespace std;
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms T(x) = x*(1+0.5*x)
// für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2;
    interval h1,h2, half=interval(0.5);
    abs_mulh1(half,0,Xi,0,h1,del1);  // abs. Fehler von 0.5*x
    abs_1px_delh(h1,del1,h2,del2);   // abs. Fehler von 1+0.5*x
    rel_mulh1(Xi,0,h2,del2,h1,bnd);  // rel. Fehler von x*(1+0.5*x)
    return bnd;  // Rückgabe des relativen Auswertefehlers
} // Term
int main()
{
// Berechnung einer relativen Fehlerschranke bez. der Auswertung
// eines Terms bei hochgenauer Arithmetik.
    interval X;  // Intervall der Maschinen-Argumente des Terms
    real bnd,diam;
    while(1) // Endlosschleife
    {
        Term_Dat(X,diam);  // Eingabe von X und diam;
            // X   : Einschließung der Maschinen-Argumente des Terms
            // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
        bnd = Term_relh(Term,X,diam);  // bnd: relative Fehlerschranke
            // bei nur hochgenauer Arithmetik bei der Termauswertung.
        cout << "Relat. Fehlerschranke bei hochgenauer Arithmetik = "
             << RndUp << bnd << endl << endl;
    }
}
```

Nach dem Start des Programms `Term_relh` erhält man mit den Eingabedaten $[a, b] = [0.125, 2]$ und der Mantissenbreite $= 10^{-6}$ nach (4.15) die relative Fehlerschranke

$$\boxed{\varepsilon(T_1) = 4.310278 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [0.125, 2];$$

Anmerkungen:

1. Soll der relative Auswertefehler für einen anderen Term abgeschätzt werden, so muss die entsprechende Fehlerabschätzung lediglich in der obigen Funktion `Term(..)` neu implementiert werden. Beachten Sie bitte, dass das Programm danach natürlich neu zu übersetzen ist!

2. Die relative Fehlerschranke $\varepsilon(T_1)$ bezieht sich **nur** auf die Maschinenzahlen $x \in [0.125, 2]$. Dies ist in der Praxis jedoch keine wirkliche Einschränkung, da relative Fehlerschranken z.B. bei Standardfunktionen **nur** für alle Maschinenzahlen eines vorgegebenen Intervalls berechnet werden.

4.5.1.2 $T_2(x) = x + \frac{1}{2}x^2$

Für den zweiten Term $T_2(x) := x + \frac{1}{2}x^2$ ist nach (4.15) für alle Maschinenzahlen $x \in [0.125, 2]$ eine Oberschranke $\varepsilon(T_2)$ des relativen Auswertefehlers zu berechnen. Im Programm `Term_relh` muss dazu lediglich in der Funktion `Term()` die entsprechende Fehlerabschätzung realisiert werden:

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms T(x) = x+0.5*x^2
// für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2;
    interval h1,h2, half=interval(0.5);

    abs_mulh1(Xi,0,Xi,0,h1,del1);       // abs. Auswertefehler von x^2
    abs_mulh1(half,0,h1,del1,h2,del2);  // abs. Fehler von 0.5*x^2
    rel_addh1(Xi,0,h2,del2,h1,bnd);     // abs. Fehler von x+0.5*x^2

    return bnd; // Rückgabe des relativen Auswertefehlers
}
```

Nach dem Start des Programms `Term_relh` erhält man mit den Eingabedaten $[a, b] = [0.125, 2]$ und der Mantissenbreite $= 10^{-6}$ nach (4.15) die relative Fehlerschranke

$$\boxed{\varepsilon(T_2) = 3.330671 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [0.125, 2];$$

Anmerkungen:

1. Die relative Fehlerschranke $\varepsilon(T_2)$ bezieht sich wie im letzten Beispiel **nur** auf die Maschinenzahlen $x \in [0.125, 2] \cap S(2, 53)$.

2. Vergleicht man mit $\varepsilon(T_1)$, so erkennt man, dass das Ausklammern des Faktors x in $T_1(x)$ den relativen Auswertefehler deutlich vergrößert.

4.5.1.3 $T_3(x) = x + \frac{1}{2}x^2,\ x = u + v$

Im dritten Term $T_3(x) = x + \frac{1}{2}x^2$ soll jetzt der zweite Summand $\frac{1}{2}x^2$ in erhöhter Genauigkeit berechnet werden, indem x mit Hilfe der Funktion `Cut24()` rundungsfehlerfrei so in zwei Summanden $u, v \in S(2, 53)$ zerlegt wird, dass u mit x auf den ersten 24 Mantissenbits übereinstimmt, während alle restlichen Mantissenbits von u verschwinden. Wir schreiben daher $u = x|_{24}$, und man erhält u mit der Anweisung `u=Cut24(x)` und v berechnet sich rundungsfehlerfrei als $v = x \ominus u = x - u$. Damit erhalten wir:

$$u = x|_{24}, \quad v := x \ominus u = x - u, \quad y := \frac{1}{2}u^2, \quad z := \frac{1}{2}v \cdot (x + u)$$

$$\rightsquigarrow \quad \frac{1}{2}x^2 = y + z \tag{4.16}$$

Da von u nur die ersten 24 Bit ungleich Null sind, ist $u \odot u$ mit maximal 48 Bit ebenfalls exakt darstellbar, d.h. $u \odot u = u \cdot u$. Die Multiplikation mit $\frac{1}{2}$ erfolgt für alle $x \in [0.125, 2]$ rundungsfehlerfrei, und für das Maschinenergebnis $\widetilde{y}$ ergibt sich:

$$\widetilde{y} := 0.5 \odot u \odot u = \frac{1}{2}u^2 = y$$

Der zweite Summand z in (4.16) muss ebenfalls auf der Maschine ausgewertet werden, und für das Maschinenergebnis $\widetilde{z}$ gilt:

$$\widetilde{z} := 0.5 \odot v \odot (x \oplus u) = \frac{1}{2}v \odot (x \oplus u)$$

da auch jetzt die Multiplikation mit 0.5 rundungsfehlerfrei erfolgt. $\frac{1}{2}x^2$ wird damit approximiert durch die Summe der beiden Maschinenzahlen y und $\widetilde{z}$:

$$\frac{1}{2}x^2 \approx y + \widetilde{z} \tag{4.17}$$

Bevor wir den durch (4.17) bedingten absoluten Approximationsfehler berechnen, benötigen wir für $|v|$ noch die folgende Abschätzung:

$$|v| < 2^{-23} \cdot |x| \tag{4.18}$$

Es sei o.B.d.A.: $x > 0$. Mit $x = 1.x_1x_2\ldots x_{52}\cdot 2^e$, $\quad u = x|_{24} = 1.x_1x_2\ldots x_{23}\cdot 2^e$ und

$$0 \le x - x|_{24} = 0.00\ldots 0x_{24}x_{25}\ldots x_{52}\cdot 2^e = v$$

erhält man:

$$\begin{aligned} v &\le (2^{-24} + 2^{-25} + \ldots + 2^{-52})\cdot 2^e = 2^{-24}(1 + 2^{-1} + 2^{-2} + \ldots + 2^{-24})\cdot 2^e \\ &= 2^{-24}\cdot\frac{1-(\frac{1}{2})^{25}}{1-\frac{1}{2}}\cdot 2^e < 2^{-24}\cdot\frac{1}{1-\frac{1}{2}}\cdot 2^e = 2^{-23}\cdot 2^e \end{aligned}$$

und wegen $1\cdot 2^e \le x$ folgt daraus unmittelbar die Behauptung (4.18).

Damit können wir jetzt den durch (4.17) bedingten absoluten Approximationsfehler durch die nachfolgende Handrechnung abschätzen. Es soll aber schon hier ausdrücklich betont werden, dass das Produkt $\frac{1}{2}x^2$ auf dem Rechner natürlich durch die auf der Maschine berechnete Summe $y \oplus \widetilde{z}$ und nicht durch die exakte Summe $y + \widetilde{z}$ approximiert wird.

$$\begin{aligned} \left|\frac{1}{2}x^2 - (y+\widetilde{z})\right| &= \left|\frac{1}{2}x^2 - [\frac{1}{2}u^2 + \frac{1}{2}v \odot (x \oplus u)]\right| \\ &= \frac{1}{2}\left|x^2 - [u^2 + (x-u)\odot(x\oplus u)]\right| \\ &= \frac{1}{2}\left|x^2 - [u^2 + (x-u)\odot(x+u)\cdot(1+\varepsilon_1)]\right| \\ &= \frac{1}{2}\left|x^2 - [u^2 + (x-u)\cdot(x+u)\cdot(1+\varepsilon_1)\cdot(1+\varepsilon_2)]\right| \end{aligned}$$

Bei nur hochgenauer Arithmetik gilt $|\varepsilon_i| \le \varepsilon := \varepsilon(h) = 2^{-52}$, und das obige Produkt $(1+\varepsilon_1)\cdot(1+\varepsilon_2)$ kann ersetzt werden durch $(1+\varepsilon_3)$ mit folgender Abschätzung:

$$|\varepsilon_3| < 1.0001\cdot 2\,\varepsilon, \quad \varepsilon = 2^{-52} \qquad \rightsquigarrow$$

$$\begin{aligned} \left|\frac{1}{2}x^2 - (y+\widetilde{z})\right| &= \frac{1}{2}\cdot\left|\, x^2 - [u^2 + (x^2-u^2)\cdot(1+\varepsilon_3)]\,\right| \\ &= \frac{1}{2}\cdot\left|\,\varepsilon_3\cdot(x^2-u^2)\,\right| = \frac{1}{2}\cdot\left|\,\varepsilon_3\cdot(x-u)(x+u)\,\right| \\ &= \frac{1}{2}\cdot\left|\,\varepsilon_3\cdot v\cdot(x+u)\,\right| \le \frac{1}{2}\cdot\left|\,\varepsilon_3\cdot v\cdot 2x\,\right| = |\varepsilon_3\cdot v\cdot x| \\ &\le 1.0001\cdot 2\,\varepsilon\cdot 2^{-23}\cdot x^2 = 1.0001\cdot\varepsilon\cdot 2^{-21}\cdot\frac{x^2}{2} \end{aligned} \tag{4.19}$$

Zum Vergleich schätzen wir jetzt noch den absoluten Fehler ab, wenn $\frac{1}{2}x^2$ durch das einfach genaue Maschinenergebnis $\frac{1}{2}\odot x\odot x$ approximiert wird:

$$\begin{aligned} \left|\frac{1}{2}\,x^2 - \frac{1}{2}\odot x\odot x\right| &= \frac{1}{2}\cdot|x^2 - x\odot x| = \frac{1}{2}\cdot|x^2 - x^2(1+\varepsilon_1)| \\ &= \frac{1}{2}\cdot|x^2\cdot\varepsilon_1| \le \frac{1}{2}\,x^2\cdot\varepsilon, \qquad \varepsilon = 2^{-52}; \end{aligned} \tag{4.20}$$

Vergleicht man mit (4.19), so erkennt man, dass die Zerlegung $x = x|_{24} + v = u + v$ und damit die Approximation $\frac{1}{2}\,x^2 = y + z \approx y + \widetilde{z}$ durch die zwei Maschinenzahlen y und $\widetilde{z}$ eine erhöhte Genauigkeit von etwa 21 zusätzlichen binären Mantissenstellen liefert. Mit den bisherigen Bezeichnungen wird $T_3(x)$ auf der Maschine ausgewertet als: $\widetilde{T}_3(x) := x \oplus (y \oplus \widetilde{z})$, und der in (4.15) definierte relative Auswertefehler lässt sich mit dem Programm `Term_relh` von Seite 126 bestimmen, wenn man dort in der Funktion `Term()` die Fehlerabschätzung wie folgt realisiert:

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms T(x) = x+0.5*x^2
// mit dem Algorithmus: x + (y+z)
// für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2;
    interval h1,h2, half=interval(0.5),ui,vi,yi;

    ui = interval(Cut24(Inf(Xi)),Cut24(Sup(Xi)));
    vi = Xi - ui;

    abs_addh1(Xi,0,ui,0,h1,del1);       // x+u
    abs_mulh1(vi,0,h1,del1,h2,del2);    // v*(x+u)
    abs_mulh1(half,0,h2,del2,h1,del1);  // z = 0.5*v*(x+u)

    yi = 0.5*ui*ui;

    abs_addh1(yi,0,h1,del1,h2,del2);   // (y+z)
    rel_addh1(Xi,0,h2,del2,yi,bnd);    // x + (y+z)

    return bnd; // Rückgabe des relativen Auswertefehlers
}
```

Nach dem Start des Programms `Term_relh` erhält man mit den Eingabedaten $[a, b] = [0.125, 2]$ und der Mantissenbreite $= 10^{-6}$ nach (4.15) die relative Fehlerschranke

$$\boxed{\varepsilon(T_3) = 3.330681 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [0.125, 2];$$

Anmerkungen:

1. Im Vergleich zu $\varepsilon(T_2) = 3.330671 \cdot 10^{-16}$ erhalten wir jetzt mit $\varepsilon(T_3)$ trotz der genaueren Darstellung von $\frac{1}{2}\,x^2 \approx y + \widetilde{z}$ einen etwas größeren Auswertefehler, da $y + \widetilde{z}$ auf der Maschine selbst wieder fehlerhaft berechnet wird, d.h. es gilt $y \oplus \widetilde{z} \neq y + \widetilde{z}$, mit $\widetilde{z} \neq z$. Erst mit dem Term $T_4(x)$ des nächsten Abschnitts werden wir durch geeignete Zusammenfassung der Summanden u, v, y, z zu einem deutlich kleineren Auswertefehler kommen.

2. Für $x \in$ `Xi` gilt: $u := x|_{24} \in$ `ui` $:=$ [`Cut24(Inf(Xi))`, `Cut24(Sup(Xi))`] und wegen $v := x - u$ erhält man für v die Einschließung: $v \in$ `Xi` $-$ `ui`.

3. Da der relative Auswertefehler nach Voraussetzung nur für Maschinenzahlen[9] $x \in [0.125, 2] \cap S(2, 53)$ zu berechnen ist und die Auswertung von $u := x|_{24}$ durch `u = Cut24(x)` rundungsfehlerfrei erfolgt, wird in `Term()` auf Seite 130 beim Funktionsaufruf `abs_addh1(Xi,0,ui,0,h1,del1)` der zweite und vierte Parameter auf Null gesetzt.

4. Da $y := \frac{1}{2}\,u^2$ auf der Maschine rundungsfehlerfrei berechnet wird und weil das Intervall `yi = 0.5*ui*ui` eine Einschließung der Maschinenzahlen y ist, wird der absolute Fehler im Funktionsaufruf `abs_addh1(yi,0,h1,del1,h2,del2)` durch den zweiten Parameter auf Null gesetzt.

4.5.1.4 $T_4(x) = x + \frac{1}{2}\,x^2,\ x = u + v$

Beim vierten Term $T_4(x) = x + \frac{1}{2}\,x^2$ wird wie bei $T_3(x)$ die Maschinenzahl x durch $u = x|_{24} =$ `Cut24(x)` und $v = x \ominus u = x - u$ rundungsfehlerfrei in die Summe $x = u + v$ zerlegt. Daraus ergeben sich mit $y = \frac{1}{2}\,u^2$ und $z = \frac{1}{2} \cdot v \cdot (x + u)$ die Beziehungen:

$$x + \frac{1}{2}\,x^2 = (u + v) + (y + z), \quad \text{d.h.} \quad \frac{1}{2}\,x^2 = y + z;$$

Bei der Auswertung von $T_4(x) = x + \frac{1}{2}\,x^2$ benutzen wir jetzt den Algorithmus:

$$\widetilde{T}_4(x) = (u \oplus y) \oplus (v \oplus \widetilde{z}) \tag{4.21}$$

Der Vorteil der rechten Seite von (4.21) besteht nun darin, dass der erste Summand $(u \oplus y)$ für $x \in [2^{-4}, 2^{30.5})$ rundungsfehlerfrei ausgewertet wird und das Maximum des absoluten Auswertefehlers des zweiten Summanden $(v \oplus \widetilde{z})$ mit $4.03 \cdot 10^{-20}$ sehr klein ausfällt. Wir erhalten damit

$$\widetilde{T}_4(x) = (u + y) \oplus (v \oplus \widetilde{z}) \tag{4.22}$$

[9] Auf Seite 124 hatten wir deshalb $\varepsilon(x) = \Delta(x) = 0$ vorausgesetzt!

und der relative Auswertefehler ist jetzt im Wesentlichen gegeben durch die relative Fehlerschranke $\varepsilon(h) = 2^{-52} = 2.2204460\ldots \cdot 10^{-16}$ der Addition bei hochgenauer Arithmetik. Der in (4.15) definierte relative Auswertefehler lässt sich jetzt wieder mit dem Programm `Term_relh` von Seite 126 bestimmen, wenn man dort in der Funktion `Term()` die Fehlerabschätzung wie folgt realisiert:

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms T(x) = x+0.5*x^2
// mit dem Algorithmus: (u+y) + (v+z)
// für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2;
    interval h1,h2, half=interval(0.5),ui,vi,yi;

    ui = interval(Cut24(Inf(Xi)),Cut24(Sup(Xi)));
    vi = Xi - ui;

    abs_addh1(Xi,0,ui,0,h1,del1);       // x+u
    abs_mulh1(vi,0,h1,del1,h2,del2);    // v*(x+u)
    abs_mulh1(half,0,h2,del2,h1,del1);  // z = 0.5*v*(x+u)

    yi = 0.5*ui*ui;  // Einschließung von y = 0.5*u*u;

    abs_addh1(h1,del1,vi,0,h2,del2);   // z+v
    h1 = yi+ui; // Einschließung für:  y+u
    del1 = 0;   // u+0.5*u^2=u+y wird bzgl. Cut24() rundungsfehlerfrei
                // addiert für x aus ( 2^(-4),2^(+30.5) )
    rel_addh1(h1,del1,h2,del2,yi,bnd);

    return bnd; // Rückgabe des relativen Auswertefehlers
}
```

Nach dem Start des Programms `Term_relh` erhält man mit den Eingabedaten $[a, b] = [0.125, 2]$ und der Mantissenbreite 10^{-6} nach (4.15) die relative Fehlerschranke

$$\boxed{\varepsilon(T_4) = 2.220462 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [0.125, 2];$$

Die Zusammenfassung der Summanden u, v, y, z nach (4.22) liefert damit eine um etwa 33% kleinere Fehlerschranke als bei $\widetilde{T}_3(x) = x \oplus (y \oplus \widetilde{z})$. Beachten Sie bitte, dass die Fehlerschranke $\varepsilon(T_4)$ als optimal angesehen werden kann, da die relative Fehlerschranke der Addition $\varepsilon(h) = 2^{-52} = 2.22044\ldots \cdot 10^{-16}$ bei hochgenauer

Arithmetik und bei Auswertung von $x + \frac{1}{2}x^2$ grundsätzlich nicht unterschritten werden kann!

Anmerkungen:

1. Dass die Auswertung von $(u \oplus y)$ für $x \in [2^{-4}, 2^{30.5})$ rundungsfehlerfrei erfolgt, kann nicht mit den in Tabelle 2.4 auf Seite 40 zusammengestellten Funktionen ermittelt werden, da bei diesen Funktionen vorausgesetzt wird, dass u und y unabhängige Größen sind. In unserem Fall sind jedoch u und y wegen $u = x|_{24}$ und $y = \frac{1}{2}u^2$ vielmehr abhängige Größen, und erst durch diese Abhängigkeit kann die rundungsfehlerfreie Addition $u \oplus y = u + y$ gezeigt werden!

2. Für den Bereich $x \in [2^{-4}, 2^{-3})$ beweisen wir jetzt, dass $u \oplus y = u + y$ erfüllt ist. Im ungünstigsten Fall gilt:

$$u = \underbrace{1.11\ldots1}_{24}\,\underbrace{00\ldots00}_{29}\cdot 2^{-4}, \qquad u^2 = \underbrace{11.1\ldots}_{48}\,00000\cdot 2^{-8} \quad \rightsquigarrow$$

$$\begin{aligned} u &= \overbrace{1.11\ldots1}^{24}\,\overbrace{00\ldots00}^{29}\cdot 2^{-4} \\ \frac{1}{2}u^2 &= 0.000\,\underbrace{11\ldots\ldots10}_{48}\cdot 2^{-4} \end{aligned}$$

   Da das letzte Mantissen-Bit von $\frac{1}{2}u^2$ gleich Null ist, wird die Summe $u \oplus \frac{1}{2}u^2$ auf der Maschine trotz Übertrag stets exakt berechnet: $u \oplus \frac{1}{2}u^2 = u + \frac{1}{2}u^2$. Für die anderen Teilintervalle $x \in [2^{-3}, 2^{-2})$, usw. zeigt man die rundungsfehlerfreie Addition ganz analog. Wir können damit die Fehlerabschätzung für den Term $T_4(x)$ auf das Intervall $[2^{-4}, 2]$ ausdehnen und erhalten mit der Mantissenbreite 10^{-6} für den relativen Auswertefehler die Fehlerschranke

$$\boxed{\varepsilon(T_4) = 2.220462 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [0.0625, 2];$$

Will man jetzt für den gleichen Term $T(x) = x + \frac{1}{2}x^2$ eine Oberschranke des relativen Auswertefehlers für alle Maschinenzahlen x eines anderen Intervalls, z.B. für alle $x \in [10^{-8}, 0.0625]$ berechnen, so ist die Anwendung des in $T_4(x)$ verwendeten Algorithmus für das jetzige Intervall nicht mehr sinnvoll, da der erste Summand $(u + v)$ in $[10^{-8}, 0.0625]$ nicht mehr rundungsfehlerfrei ausgewertet werden kann, was etwa zur Verdoppelung des Auswertefehlers führen würde. Einen viel besseren Wert erhält man, wenn der in $T_2(x)$ verwendete Algorithmus aus Abschnitt 4.5.1.2 zur Anwendung kommt. Startet man das Programm `Term_relh` mit der auf Seite 127 angegebenen Funktion `real Term()`, so erhält man mit der Mantissenbreite 10^{-6} für den relativen Auswertefehler die Oberschranke

$$\boxed{\varepsilon(T_2) = 2.287733 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in [10^{-8}, 0.0625];$$

4.5.2 $x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3$

Der Term $T(x) = x + \frac{1}{2}\cdot x^2 + \frac{1}{6}\,x^3$ ist für alle Maschinenzahlen $x \in [10^{-17}, 0.223144]$ auszuwerten, und für den dabei auftretenden relativen Auswertefehler ist eine garantierte Oberschranke $\varepsilon(T)$ zu berechnen:

$$\left|\frac{T(x) - \widetilde{T}(x)}{T(x)}\right| \leq \varepsilon(T), \quad x \in [a,b] = [10^{-17}, 0.223144] \tag{4.23}$$

$\widetilde{T}(x)$ ist dabei das i.a. fehlerbehaftete Maschinenergebnis, wenn $T(x)$ in hochgenauer Arithmetik ausgewertet wird. Wie im Abschnitt 4.5.1.3 wird die Maschinenzahl x durch $u = x|_{24} =$ `Cut24(x)` und $v = x \ominus u = x - u$ rundungsfehlerfrei in die Summe $x = u + v$ zerlegt. Daraus ergeben sich mit $y = \frac{1}{2}\,u^2, z = \frac{1}{2}\cdot v\cdot(x+u)$ und $q = \frac{1}{6}\,x^3$ die Beziehungen:

$$x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3 = u + v + y + z + q, \quad \text{mit} \quad \frac{1}{2}\,x^2 = y + z, \quad q = \frac{1}{6}\,x^3; \tag{4.24}$$

Um möglichst kleine Auswertefehler zu erhalten, wird das vorgegebene Intervall $I = [10^{-17}, 0.223144]$ in zwei Teilintervalle $I_1 = [10^{-17}, 2^{-4}]$ und $I_2 = [2^{-4}, 0.223144]$ zerlegt, und für jedes Teilintervall wird ein eigener Algorithmus angegeben, mit dem die Summanden rechts in (4.24) in geeigneter Weise zusammengefasst werden:

$$T_1(x) = x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3; \qquad \widetilde{T}_1(x) = x \oplus (y \oplus (\widetilde{q} \oplus \widetilde{z})), \qquad x \in I_1 \tag{4.25}$$

$$T_2(x) = x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3; \qquad \widetilde{T}_2(x) = (u \oplus y) \oplus (\widetilde{q} \oplus (v \oplus \widetilde{z})), \quad x \in I_2 \tag{4.26}$$

$\widetilde{z}$ und $\widetilde{q}$ sind die fehlerbehafteten Maschinenergebnisse bei Auswertung von z und q:

$$\begin{aligned} \widetilde{z} &= 0.5 \cdot v \odot (x \oplus u) \\ \widetilde{q} &= (1 \oslash 6) \odot x \odot x \odot x \end{aligned}$$

Mit der Funktion `frac_tools()` aus dem Modul `bnd_util` wird durch den Aufruf

```
frac_tools(1,6,u6,Next,del6,eps6);
```

für den Bruch $\frac{1}{6}$ mit `u6` eine optimale Einschließung und mit `Next` die zu $\frac{1}{6}$ nächstgelegene Rasterzahl im double-Format berechnet. Der bei dieser Rundung aufgetretene absolute und relative Fehler wird durch die berechneten Oberschranken `del6` bzw. `eps6` abgeschätzt. Für die Fehlerabschätzung wird die Einschließung `u6` und die absolute Fehlerschranke `del6` benötigt, dabei wird vorausgesetzt, dass bei der Auswertung von $T(x)$ die Multiplikation mit $\frac{1}{6}$ durch die Multiplikation mit der zu $\frac{1}{6}$ nächstgelegenen Rasterzahl `Next` erfolgt.

Beachten Sie bitte, dass die in (4.25) und (4.26) angegebenen Algorithmen zur Auswertung von $T(x)$ keine zeitaufwendige doppelte Präzision[10] benutzen, sondern lediglich darauf beruhen, dass die `real`-Summanden u, v, x, y, z, q in den Teilintervallen I_1 und I_2 in geeigneter Reihenfolge zusammengefasst werden, um den links in (4.23) definierten relativen Auswertefehler möglichst klein zu halten. Der Haupttrick ist dabei die rundungsfehlerfreie Zerlegung von x in die Summe $x = u + v$, wobei nach Seite 134 die Summe $u + y$ für $x \in [2^{-4}, 2^{30.5})$ auf der Maschine rundungsfehlerfrei ausgewertet werden kann, d.h. $u \oplus y = u + y$, mit $y = \frac{1}{2}u^2$ und $u = x|_{24} =$ `Cut24(x)`.

4.5.2.1 $T_1(x) = x + (y + (q + z))$

Die Auswertung des Terms $T(x) = x + \frac{1}{2}x^2 + \frac{1}{6}x^3$ erfolgt im Intervall $I_1 = [10^{-17}, 2^{-4}]$ mit Hilfe von $\widetilde{T}_1(x) = x \oplus (y \oplus (\widetilde{q} \oplus \widetilde{z}))$, wobei $y, \widetilde{q}, \widetilde{z}$ wie auf Seite 135 definiert sind.

$$\left|\frac{T(x) - \widetilde{T}_1(x)}{T(x)}\right| \le \varepsilon(T_1), \qquad x \in I_1 = [10^{-17}, 2^{-4}] \tag{4.27}$$

Eine garantierte Oberschranke $\varepsilon(T_1)$ des relativen Auswertefehlers erhält man mit dem Programm `Term_relh` von Seite 126, wenn man dort in der Funktion `Term()` die Fehlerabschätzung wie folgt realisiert:

```
real Term_1(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x)   = x + 0.5*x^2 + (1/6)*x^3 mit dem Algorithmus:
// T_1(x) = x + (y+(q+z)); x=u+v, y = 0.5*u^2, z = 0.5*v*(x+u),
// q=(1/6)*x^3 für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2,del3,del6,eps6,Next;
    interval h1,h2,h3,u6, half=interval(0.5),ui,vi,yi;

    ui = interval(Cut24(Inf(Xi)),Cut24(Sup(Xi)));
    vi = Xi - ui;

    abs_addh1(Xi,0,ui,0,h1,del1);       // x+u
    abs_mulh1(vi,0,h1,del1,h2,del2);    // v*(x+u)
    abs_mulh1(half,0,h2,del2,h1,del1);  // z = 0.5*v*(x+u)
```

[10] Die Präzision einer Rechnung ist die Anzahl der Mantissenbits der verwendeten Variablen. Die Präzision einer `real`-Variablen ist damit 53. Die Präzision einer Rechnung ist nicht zu verwechseln mit der Ergebnisgenauigkeit, die durch den absoluten oder relativen Fehler bestimmt ist.

```
    frac_tools(1,6,u6,Next,del6,eps6);
    // u6: Einschließung von 1/6;  del6: absolute Fehlerschranke
    abs_mulh1(Xi,0,Xi,0,h2,del2);        // x*x
    abs_mulh1(h2,del2,Xi,0,h3,del3);     // x^3
    abs_mulh1(h1,del1,u6,del6,h2,del2);  // q = (1/6)*x^3

    abs_addh1(h1,del1,h2,del2,h3,del3);  // q+z
    yi = 0.5*ui*ui;  // Einschließung von y = 0.5*u*u;
    abs_addh1(yi,0,h3,del3,h1,del1);     // y+(q+z)
    rel_addh1(Xi,0,h1,del1,yi,bnd);      // x + (y+(q+z))

    return bnd; // Rückgabe des relativen Auswertefehlers
}
```

Nach dem Start des Programms `Term_relh` von Seite 126 erhält man mit den Eingabedaten $[a, b] = [\texttt{1e} - \texttt{17}, \texttt{0.0625}]$ und der relativen Mantissenbreite $= 10^{-5}$ nach (4.27) die relative Fehlerschranke

$$\boxed{\varepsilon(T_1) = 2.287742 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in I_1 = [10^{-17}, 2^{-4}];$$

Anmerkungen:

1. Die Auswertung des Terms $T(x) = x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3$ liefert in der Umgebung des Ursprungs eine grobe Näherung für die Funktion $f(x) = e^x - 1$. Für eine höhere Genauigkeit müssen natürlich weitere Taylorsummanden berücksichtigt werden. An Stelle von $\frac{1}{6}\,x^3$ wird dann das Polynom $x^3(\frac{1}{3!} + \frac{1}{4!}\,x^1 + \ldots)$ nach dem Hornerschema ausgewertet, wobei die Zerlegung von $x = u+v$ hier wegen der viel kleineren Summanden nicht mehr notwendig ist. In [11] findet man zur Implementierung von $e^x - 1$ nach dem Tabellenverfahren [27] den vollständigen Algorithmus und die dazugehörige Fehlerabschätzung.

2. Wendet man den gleichen, durch $T_1(x)$ vorgegebenen Algorithmus auch auf das Intervall $I_2 = [2^{-4}, 0.223144]$ an, so erhält man mit dem Programm `Term_relh` für den relativen Auswertefehler die größere Oberschranke $2.443352 \cdot 10^{-16}$. Eine deutlich kleinere Oberschranke ergibt sich mit Hilfe des in (4.26) auf Seite 135 angegebenen Terms $\widetilde{T}_2(x) = (u \oplus y) \oplus (\widetilde{q} \oplus (v \oplus \widetilde{z}))$. Der hier verwendete Algorithmus benutzt die Zerlegung $x = u + v$, wobei die Summe $u + y$ auf der Maschine rundungsfehlerfrei berechnet wird, d.h. $u \oplus y = u + y$. Die zugehörige Fehlerabschätzung findet man im folgenden Unterabschnitt.

4.5.2.2 $T_2(x) = (u + y) + (q + (v + z))$

Die Auswertung von $T(x) = x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3$ erfolgt im Intervall $I_2 = [2^{-4}, 0.223144]$ mit Hilfe von $\widetilde{T}_2(x) = (u \oplus y) \oplus (\widetilde{q} \oplus (v \oplus \widetilde{z}))$, wobei $u, v, y, \widetilde{q}, \widetilde{z}$ wie auf Seite 135 definiert sind.

$$\left|\frac{T(x) - \widetilde{T}_2(x)}{T(x)}\right| \leq \varepsilon(T_2), \qquad x \in I_2 = [2^{-4}, 0.223144] \tag{4.28}$$

Eine garantierte Oberschranke $\varepsilon(T_2)$ des relativen Auswertefehlers erhält man mit dem Programm `Term_relh` von Seite 126, wenn man dort in der Funktion `Term()` die Fehlerabschätzung wie folgt realisiert:

```
real Term_2(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x) = x + 0.5*x^2 + (1/6)*x^3 mit dem Algorithmus:
// T(x) = (u+y) + (q+(v+z));  x=u+v, y = 0.5*u^2, z = 0.5*v*(x+u);
// für alle Maschinenzahlen des Teilintervalls Xi.
{
    real bnd,del1,del2,del3,Next,del6,eps6;
    interval h1,h2,h3,u6, half=interval(0.5),ui,vi,yi;
    ui = interval(Cut24(Inf(Xi)),Cut24(Sup(Xi)));
    vi = Xi - ui;

    abs_addh1(Xi,0,ui,0,h1,del1);       // x+u
    abs_mulh1(vi,0,h1,del1,h2,del2);    // v*(x+u)
    abs_mulh1(half,0,h2,del2,h1,del1);  // z = 0.5*v*(x+u)
    abs_addh1(h1,del1,vi,0,h2,del2);    // v+z
    frac_tools(1,6,u6,Next,del6,eps6);
    // u6: Einschließung von 1/6;  del6: absolute Fehlerschranke
    abs_mulh1(Xi,0,Xi,0,h1,del1);        // x*x
    abs_mulh1(h1,del1,Xi,0,h3,del3);     // x^3
    abs_mulh1(h3,del3,u6,del6,h1,del1);  // q = (1/6)*x^3
    abs_addh1(h1,del1,h2,del2,h3,del3);  // q+(v+z)

    yi = 0.5*ui*ui;  // Einschließung von y = 0.5*u*u;
    h1 = yi+ui; // Einschließung für:  y+u
    del1 = 0;   // u+0.5*u^2=u+y wird bez. Cut24() rundungsfehlerfrei
                // addiert für x aus [ 2^(-4),2^(+30.5) )
    rel_addh1(h1,del1,h3,del3,yi,bnd);
    return bnd; // Rückgabe des relativen Auswertefehlers
}
```

Nach dem Start des **C-XSC** Programms `Term_relh` erhält man mit den Eingabedaten $[a, b] = [0.0625, 0.223144]$ und der relativen Mantissenbreite 10^{-5} nach (4.28) die relative Fehlerschranke

$$\boxed{\varepsilon(T_2) = 2.290450 \cdot 10^{-16}} \quad \text{für alle Maschinenzahlen } x \in I_2 = [2^{-4}, 0.223144];$$

Anmerkungen:

1. Mit der gleichen relativen Mantissenbreite 10^{-5} und den Argumentintervallen $I_3 = [0.125, 0.223144]$ und $I_4 = [0.223143, 0.223144]$ erhält man mit dem Programm `Term_relh` die gleiche Oberschranke $\varepsilon(T_2) = 2.290450 \cdot 10^{-16}$. Man kann daher vermuten, dass das gesuchte Maximum des relativen Auswertefehlers an der gemeinsamen oberen Intervallgrenze angenommen wird.

2. In [11] wird für das obige Intervall $I_3 = [0.125, 0.223144]$ auf Seite 20 (Bereich IId) für den relativen Gesamtfehler der Funktion $e^x - 1$ die im Vergleich zu $\varepsilon(T_2)$ größere Schranke $2.359\ldots \cdot 10^{-16}$ angegeben, da dort der Auswertefehler des Approximationspolynoms und der durch das Abschneiden der Taylorreihe bedingte Approximationsfehler zusätzlich zu berücksichtigen ist.

3. Die bisherigen Beispiele zeigen, dass die Berechnung einer möglichst kleinen Oberschranke für den relativen Auswertefehler eines vorgegebenen Terms $T(x)$ für alle Maschinenzahlen eines Intervalls $[a, b]$ nicht durch ein Patentrezept gelöst werden kann, sondern dass jeder Term sogar in einzelnen Teilintervallen jeweils eigene Algorithmen erfordert, in denen durch die Zerlegung von x in zwei geeignete Summanden $x = u + v$ meist eine höhere Präzision simuliert wird. Dabei wird diese höhere Präzision i.a. mit einer etwas größeren Laufzeit erkauft.

4. Im folgenden Kapitel wird ein sehr einfaches Fehlerkalkül vorgestellt, das in speziellen Situationen eine Oberschranke des relativen Fehlers sehr einfach liefert, wobei jedoch die in diesem Kapitel ausführlich beschriebenen Optimierungsmöglichkeiten nicht zur Verfügung stehen. Die Grundlagen dieses Fehlerkalküls wurden gelegt in den Arbeiten von [19],[6] und beziehen sich auf alle vier Grundoperationen. Im folgenden Kapitel 5 beschränken wir uns jedoch auf die einfachsten Operationen Multiplikation und Division, die in der Praxis in den angesprochenen speziellen Situationen am häufigsten auftreten.

Kapitel 5

Ein einfaches, nicht optimiertes Fehlerkalkül

5.1 Problemstellung

Es gibt spezielle Situationen, in denen von zwei oder auch mehreren Ausdrücken die relativen Fehler schon bekannt sind und abschließend nur noch der relative Fehler des Produktes oder des Quotienten der Ausdrücke gesucht ist, dabei setzen wir auch jetzt voraus, dass die Grundoperationen nur in hochgenauer Arithmetik erfolgen, d.h. mit der relativen Fehlerschranke $\varepsilon(h) := 2^{-52}$. Während die bekannten relativen Fehlerschranken der vorliegenden Ausdrücke durchaus mit den Methoden des letzten Kapitels optimiert sein können, wird auf eine solche Optimierung der relativen Fehlerschranken bei der Berechnung der genannten Produkte oder Quotienten mit diesem Fehlerkalkül verzichtet. Wir betrachten ein einfaches Beispiel:

Der fehlerbehaftete Maschinenwert $\widetilde{a} = a \cdot (1+\varepsilon_a) \in S(2,53)$ ist mit dem exakten Maschinenwert $b \in S(2,53)$ zu multiplizieren. Liegt das Maschinenprodukt $\widetilde{a} \odot b$ im normalisierten Bereich, so gilt

$$\begin{aligned} \widetilde{a} \odot b &= [a \cdot (1+\varepsilon_a)] \odot b = (a \cdot b) \cdot (1+\varepsilon_a)(1+e_h) \\ &= (a \cdot b) \cdot (1+\varepsilon_2); \quad |\varepsilon_a| \leq \varepsilon(a), \quad |\varepsilon_h| \leq \varepsilon(h); \end{aligned}$$

Um für das Produkt $\widetilde{a} \odot b$ den relativen Fehler ε_2 abzuschätzen, benötigt man im Ausdruck $(1+\varepsilon_a)(1+e_h) = (1+\varepsilon_2)$ eine Abschätzung für $|\varepsilon_2| \leq \varepsilon(2) = ?$ unter den Bedingungen: $|\varepsilon_a| \leq \varepsilon(a)$ und $|\varepsilon_h| \leq \varepsilon(h)$. Das Problem wird gelöst durch eine entsprechende Extremwertaufgabe einer Funktion zweier Variabler. Man findet durch elementare Rechnungen

$$|\varepsilon_2| \leq \varepsilon(a) + \varepsilon(h) + \varepsilon(a) \cdot \varepsilon(h) = \varepsilon(2) \tag{5.1}$$

Bei der Auswertung der rechten Seite in (5.1) ist darauf zu achten, dass die auftretenden Zwischen-Summen und Produkte jeweils zur nächst größeren Rasterzahl zu runden sind.

5.2 Funktionen aus `abs_relh` zur Abschätzung rel. Fehler

In folgender Tabelle sind die Funktionen aus dem Modul `abs_relh` zusammengestellt, mit denen man zu vorgegebenen Fehlertermen eine Oberschranke des relativen Gesamtfehlers berechnen kann.

5.3 Anwendungsbeispiele

1. Beispiel

Wir kommen auf das Beispiel von Seite 141 zurück und wählen $\varepsilon(a) = 1.11 \cdot 10^{-16}$ und $\varepsilon(h) =$ `eps_h` $= 2^{-52}$ wegen der verlangten hochgenauen Arithmetik bei der Multiplikation. Es muss darauf geachtet werden, dass der dezimale Wert für $\varepsilon(a)$ i.a. keine Maschinenzahl ist und daher zur nächst größeren Maschinenzahl `eps_a` $\geq 1.11 \cdot 10^{-16}$ zu runden ist. Eine interaktive Lösung dieses Problems, bei dem der Anwender den dezimalen Wert über die Tastatur eingeben muss, zeigt das folgende kleine **C-XSC** Programm.

```
#include <iostream>      // Wegen cout
#include "abs_relh.hpp"  // Wegen eps_2()

using namespace std;
using namespace cxsc;

int main(){
    const real eps_h = comp(0.5,-51);  // eps_h = 2^(-52)
    real S,eps_a;
    cout << "eps(a) = ?" << endl;  cin >> RndUp >> eps_a;
    S = eps_2(eps_a,eps_h);
    cout << RndUp << "S = " << S << endl;
    return 0;
}
```

Mit dem Eingabewert `1.11e-16` für `eps(a)` liefert das obige Programm die Oberschranke des relativen Fehlers: $|\varepsilon_2| \leq$ `S` $= 3.330447 \cdot 10^{-16}$.

2. Beispiel

Wir wollen jetzt die beiden fehlerbehafteten Maschinenzahlen $\widetilde{a} = a \cdot (1 + \varepsilon_a)$ und $\widetilde{b} = b \cdot (1 + \varepsilon_b)$ multiplizieren. Liegt das Maschinenprodukt $\widetilde{a} \odot \widetilde{b}$ im normalisierten Bereich, so gilt

$$\begin{aligned} \widetilde{a} \odot \widetilde{b} &= [a \cdot (1 + \varepsilon_a)] \odot [b \cdot (1 + \varepsilon_b)] = (a \cdot b) \cdot (1 + \varepsilon_a)(1 + \varepsilon_b)(1 + e_h) \\ &= (a \cdot b) \cdot (1 + \varepsilon_3); \quad |\varepsilon_a| \leq \varepsilon(a), \quad |\varepsilon_b| \leq \varepsilon(b)|, \quad |\varepsilon_h| \leq \varepsilon(h); \end{aligned}$$

Mit dem Eingabewerten `1.11e-16` für `eps(a)` und `2.00e-16` für `eps(b)`liefert das folgende Programm

```
#include <iostream>      // Wegen cout
#include "abs_relh.hpp"  // Wegen eps_3()

using namespace std;
using namespace cxsc;

int main(){
    const real eps_h = comp(0.5,-51);  // eps_h = 2^(-52)
    real S,eps_a,eps_b;
    cout << "eps(a) = ?" << endl;  cin >> RndUp >> eps_a;
    cout << "eps(b) = ?" << endl;  cin >> RndUp >> eps_b;

    S = eps_3(eps_a,eps_b,eps_h);
    cout << RndUp << "S = " << S << endl;
    return 0;
}
```

die Oberschranke `S` des relativen Fehlers: $|\varepsilon_3| \leq \mathtt{S} = 5.330447 \cdot 10^{-16}$.

3. Beispiel

Im letzten Beispiel soll der Quotient der beiden fehlerbehafteten Maschinenzahlen $\widetilde{Z} = Z \cdot (1+\varepsilon_Z)$ und $\widetilde{N} = N \cdot (1+\varepsilon_N)$ berechnet werden. Liegt das Maschinenergebnis $\widetilde{Z} \oslash \widetilde{N}$ im normalisierten Bereich, so gilt

$$\begin{aligned} \widetilde{Z} \oslash \widetilde{N} &= Z \cdot (1+\varepsilon_Z) \oslash N \cdot (1+\varepsilon_N) = \frac{Z}{N} \cdot \frac{(1+\varepsilon_Z)(1+\varepsilon_h)}{(1+\varepsilon_N)} \\ &= \frac{Z}{N} \cdot (1+\varepsilon_q); \quad |\varepsilon_Z| \leq \varepsilon(Z), \quad |\varepsilon_N| \leq \varepsilon(N)|, \quad |\varepsilon_h| \leq \varepsilon(h) = 2^{-52}; \end{aligned}$$

Das Programm

```
#include <iostream>      // Wegen cout
#include "abs_relh.hpp"  // Wegen eps_ZdN()

using namespace std;
using namespace cxsc;

int main(){
    real S,eps_Z,eps_N;
    cout << "eps(Z) = ?" << endl;  cin >> RndUp >> eps_Z;
    cout << "eps(N) = ?" << endl;  cin >> RndUp >> eps_N;
```

```
    S = eps_ZdN(eps_Z,eps_N);
    cout << RndUp << "S = " << S << endl;
    return 0;
}
```

liefert mit den Eingabewerten `1.11e-16` für `eps(Z)` und `2.00e-16` für `eps(N)` die Oberschranke `S` des relativen Fehlers: $|\varepsilon_q| \leq \mathtt{S} = 5.330447 \cdot 10^{-16}$.

Anmerkungen:

1. Die auf 6 Nachkommastellen aufgerundeten Oberschranken $\mathtt{S} = 5.330447 \cdot 10^{-16}$ für $|\varepsilon_3|$ und $|\varepsilon_q|$ haben hier die gleichen Werte, tatsächlich zeigen Rechnungen in höhere Präzision, dass die Oberschranke von $|\varepsilon_q|$ minimal größer ist als die von $|\varepsilon_3|$.

2. Ein weiteres Beispiel aus der Praxis zur Anwendung dieses Fehlerkalküls findet man auf Seite 346.

Funktionen aus `abs_relh` zur Berechnung relativer Fehlerschranken	
Alle Eingangsparameter u. Rückgabewerte sind vom Typ `real, int n;`	
Fehlerterm	$(1+\varepsilon_a)(1+\varepsilon_b) = (1+\varepsilon_2)$
Bedingungen	$\lvert\varepsilon_a\rvert \le \varepsilon(a) \le$ `eps_a`, $\lvert\varepsilon_b\rvert \le \varepsilon(b) \le$ `eps_b`;
Fktn-Aufruf	`S = eps_2(eps_a,eps_b);` $\rightsquigarrow$ $\lvert\varepsilon_2\rvert \le$ `S`
Fehlerterm	$(1+\varepsilon_a)(1+\varepsilon_b)(1+\varepsilon_c) = (1+\varepsilon_3)$
Bedingungen	$\lvert\varepsilon_a\rvert \le \varepsilon(a) \le$ `eps_a`, $\lvert\varepsilon_b\rvert \le \varepsilon(b) \le$ `eps_b`, ...
Fktn-Aufruf	`S = eps_3(eps_a,eps_b,eps_c);` $\rightsquigarrow$ $\lvert\varepsilon_3\rvert \le$ `S`
Fehlerterm	$(1+\varepsilon_a)(1+\varepsilon_b)(1+\varepsilon_c)(1+\varepsilon_d) = (1+\varepsilon_4)$
Bedingungen	$\lvert\varepsilon_a\rvert \le \varepsilon(a) \le$ `eps_a`, $\lvert\varepsilon_b\rvert \le \varepsilon(b) \le$ `eps_b`, ...
Fktn-Aufruf	`S = eps_4(eps_a,eps_b,eps_c,eps_d);` $\rightsquigarrow$ $\lvert\varepsilon_4\rvert \le$ `S`
Fehlerterm	$(1+\varepsilon_a)^n = (1+\varepsilon_n),\ n \ge 1$
Bedingungen	$\lvert\varepsilon_a\rvert \le \varepsilon(a) \le$ `eps_a`, $\quad n \ge 1$;
Fktn-Aufruf	`S = eps_pow(eps_a,n);` `int n;` $\rightsquigarrow$ $\lvert\varepsilon_n\rvert \le$ `S`
Fehlerterm	$1/(1+\varepsilon_a) = (1+\varepsilon_r)$,
Bedingungen	$\lvert\varepsilon_a\rvert \le \varepsilon(a) \le$ `eps_a`, $\quad$ `eps_a` ≤ 0.5;
Fktn-Aufruf	`S = eps_rezi(eps_a);` $\rightsquigarrow$ $\lvert\varepsilon_r\rvert \le$ `S`
Fehlerterm	$(1+\varepsilon_Z)(1+\varepsilon_h)/(1+\varepsilon_N) = (1+\varepsilon_q)$,
Bedingungen	$\lvert\varepsilon_Z\rvert \le \varepsilon(Z) \le$ `eps_Z`, $\lvert\varepsilon_N\rvert \le \varepsilon(N) \le$ `eps_N`, $\lvert\varepsilon_h\rvert \le 2^{-52}$
Fktn-Aufruf	`S = eps_ZdN(eps_Z,eps_N);` $\rightsquigarrow$ $\lvert\varepsilon_q\rvert \le$ `S`

Tabelle 5.1: Berechnung relativer Fehlerschranken `S`

Kapitel 6

Fehlerbehaftete Funktionsargumente

In den bisher betrachteten Termen $T(x)$ kamen bisher nur die vier Grundoperationen zur Anwendung, wobei als Operanden entweder Konstanten oder Variablen gewählt wurden. Wir wollen jetzt auch Funktionsterme als Operanden zulassen, d.h. wir betrachten z.B. den Term

$$T(x) = \sqrt{1 + x^2} + 1, \quad \text{mit} \quad x \in \mathbb{R} \cap S(2, 53),$$

der damit nur für Maschinenzahlen $\widetilde{x} \in S(2, 53)$ auszuwerten ist. Das Maschinenergebnis bezeichnen wir wieder mit:

$$\widetilde{T}(\widetilde{x}) = \widetilde{\sqrt{1 \oplus \widetilde{x} \odot \widetilde{x}}} \oplus 1,$$

wobei $\widetilde{\sqrt{\ }}$ die mit Hilfe der **C-XSC** Funktion `sqrt( )` $\approx \sqrt{\ }$ berechnete Maschinennäherung ist. Wegen $1 \oplus \widetilde{x} \odot \widetilde{x} \neq 1 + \widetilde{x}^2$ wird die Wurzelfunktion `sqrt(...)` selbst wieder mit fehlerbehafteten Argumenten aufgerufen, und die Aufgabe besteht darin, z.B. eine Oberschranke des absoluten Fehlers $|\widetilde{T}(\widetilde{x}) - T(\widetilde{x})|$ für alle Maschinenzahlen $\widetilde{x}$ eines vorgegebenen Intervalls zu berechnen. Im folgenden Unterabschnitt werden alle Formeln zur Berechnung des absoluten oder relativen Fehlers hergeleitet, wenn eine vorgegebene differenzierbare Funktion f durch eine fehlerbehaftete Funktion $\widetilde{f}$ auf der Maschine approximiert wird, wobei wir zulassen müssen, dass die exakten Funktionsargumente $a \in A$ selbst nur als fehlerbehaftete Maschinenwerte $\widetilde{a} \in \widetilde{A}$ vorliegen, [3].

Im Unterabschnitt 6.2 werden dann diese Formeln auf einige spezielle Standardfunktionen angewandt, und im Unterabschnitt 8.1 findet man eine komplette Fehlerabschätzung der **C-XSC** Funktion $\sqrt{1 + x} - 1$.

6.1 Allgemeine Fehlerabschätzung

Zu einer in ihrem Definitionsbereich D differenzierbaren Funktion $f : D \to I\!R$ betrachten wir zunächst die exakten Funktionsargumente $a \in A \subseteq D$, die durch das Maschinenintervall $A \in IR$ eingeschlossen werden. Es wird zusätzlich angenommen, dass die reellen $a \in A$ auf der Maschine fehlerhaft berechnet wurden und nur als Maschinenzahlen $\widetilde{a} \in S(2,53)$ vorliegen. Zwischen a und $\widetilde{a}$ gelten nach Voraussetzung die folgenden Beziehungen:

$$\widetilde{a} = a + \Delta_a, \quad \text{mit} \quad |\Delta_a| \leq \Delta(a) \text{ für alle } a \in A; \tag{6.1}$$

$$\widetilde{a} = a \cdot (1 + \varepsilon_a), \quad \text{mit} \quad |\varepsilon_a| \leq \varepsilon(a) \text{ für alle } a \in A; \tag{6.2}$$

Die Maschinenzahlen $\widetilde{a}$ werden dann eingeschlossen durch das Maschinenintervall $\widetilde{A} = A \diamondsuit [-\Delta(a), +\Delta(a)] \in IR$, wobei die Intervalladdition auf der Maschine durch $\diamondsuit$ symbolisiert wird.

Satz 6.1.1 Sei $f : D \to I\!R$ differenzierbar auf der Definitionsmenge

$$D \supseteq \widetilde{A} = A \diamondsuit [-\Delta(a), +\Delta(a)], \quad D \subset I\!R$$

Für die auf der Maschine implementierte Näherungsfunktion $\widetilde{f} : D \cap S \to S$ bez. f sei $\varepsilon(f)$ eine bekannte obere Schranke für den relativen Fehler im normalisierten Bereich und $\Delta(f) \geq 0$ eine bekannte obere Schranke für den absoluten Fehler im Unterlaufbereich, d.h. es gelte

$$|f(\widetilde{x}) - \widetilde{f}(\widetilde{x})| \leq \varepsilon(f) \cdot |f(\widetilde{x})| \tag{6.3}$$

für alle $\widetilde{x} \in S(2,53) = S$ mit $|f(\widetilde{x})| \in [\texttt{MinReal}, \texttt{MaxReal}]$ und

$$|f(\widetilde{x}) - \widetilde{f}(\widetilde{x})| \leq \Delta(f) \tag{6.4}$$

für alle $\widetilde{x} \in S$ mit $|f(\widetilde{x})| \in [0, \texttt{MinReal})$. Mit diesen Voraussetzungen und mit

$$\Delta_{dat,f} = \begin{cases} \Delta(a) \cdot |f'(a + [-\Delta(a), \Delta(a)])|\,, & \text{falls } \Delta(a)\text{gegeben ist,} \\ |a|\,\varepsilon(a) \cdot |f'(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)])|\,, & \text{falls } \varepsilon(a)\text{gegeben ist,} \end{cases} \tag{6.5}$$

erhält man für den absoluten Fehler die Abschätzung:

$$\begin{aligned} \left| f(a) - \widetilde{f}(\widetilde{a}) \right| &\leq \Delta_{dat,f} + \\ &+ \begin{cases} \Delta(f), & \text{falls } |f(\widetilde{a})| \in [0, \texttt{MinReal}) \\ \varepsilon(f)\,(\Delta_{dat,f} + |f(a)|)\,, & \text{falls } |f(\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \end{cases} \end{aligned} \tag{6.6}$$

Beweis: Seien $a \in A$ und $\widetilde{a} \in \widetilde{A} \cap S$ beliebig und $\Delta_a = \widetilde{a} - a \in [-\Delta(a), \Delta(a)]$.

1. **Fortgepflanzter Datenfehler:**
   Aus dem Mittelwertsatz der Differentialrechnung folgt

$$f(\widetilde{a}) = f(a + \Delta_a) = f(a) + f'(a + \theta \cdot \Delta_a) \cdot \Delta_a, \quad \text{mit } \theta \in (0,1) \quad \rightsquigarrow$$
$$\begin{aligned} |f(a) - f(\widetilde{a})| &= \big|\Delta_a| \cdot |f'(a + \theta \cdot \Delta_a)\big| \\ &\leq \Delta(a) \cdot \big|f'\big(a + [-\Delta(a), \Delta(a)]\big)\big| = \Delta_{dat,f}, \text{ gegeben: } \Delta(a); \end{aligned}$$

   Mit $\Delta_a = \varepsilon_a \cdot a$ erhält man bei gegebenem $\varepsilon(a)$:

$$\begin{aligned} |f(a) - f(\widetilde{a})| &= \big|a \cdot \varepsilon_a| \cdot |f'(a + \theta \cdot \varepsilon_a \cdot a)\big| \\ &\leq |a| \cdot \varepsilon(a) \cdot \big|f'\big(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\big)\big| = \Delta_{dat,f} \end{aligned}$$

2. **Rundungsfehler:**
   Nach (6.3) und (6.4) gilt

$$\Big|f(\widetilde{a}) - \widetilde{f}(\widetilde{a})\Big| \leq \begin{cases} \Delta(f), & \text{falls } |f(\widetilde{a})| \in [\texttt{0}, \texttt{MinReal}), \\ \varepsilon(f) \cdot |f(\widetilde{a})|, & \text{falls } |f(\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \end{cases}$$

   Wegen $|f(\widetilde{a})| = |(f(\widetilde{a}) - f(a)) + f(a)| \leq |f(\widetilde{a}) - f(a)| + |f(a)| = \Delta_{dat,f} + |f(a)|$ erhält man

$$\Big|f(\widetilde{a}) - \widetilde{f}(\widetilde{a})\Big| \leq \begin{cases} \Delta(f), & \text{falls } |f(\widetilde{a})| \in [\texttt{0}, \texttt{MinReal}), \\ \varepsilon(f) \cdot (\Delta_{dat,f} + |f(a)|), & \text{falls } |f(\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \end{cases}$$

3. **Gesamtfehler:**
   Da $a \in A$ und $\widetilde{a} \in \widetilde{A} \cap S$ beliebig waren, folgt die Behauptung des Satzes jetzt aus der Dreiecksungleichung, angewandt auf den Gesamtfehler:

$$\begin{aligned} \Big|f(a) - \widetilde{f}(\widetilde{a})\Big| &\leq |f(a) - f(\widetilde{a})| + \Big|f(\widetilde{a}) - \widetilde{f}(\widetilde{a})\Big| \\ &\leq \Delta_{dat,f} + \\ &\quad + \begin{cases} \Delta(f), & \text{falls } |f(\widetilde{a})| \in [\texttt{0}, \texttt{MinReal}), \\ \varepsilon(f) \cdot (\Delta_{dat,f} + |f(a)|), & \text{falls } |f(\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \end{cases} \end{aligned}$$

Satz 6.1.2 Für den relativen Gesamtfehler gilt nach Satz 6.1.1 mit

$$\varepsilon_{dat,f} = \begin{cases} \Delta(a) \cdot \left|\dfrac{f'(a + [-\Delta(a), \Delta(a)])}{f(a)}\right|, & \text{falls } \Delta(a) \text{ gegeben ist,} \\ |a|\,\varepsilon(a) \cdot \left|\dfrac{f'\big(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\big)}{f(a)}\right|, & \text{falls } \varepsilon(a) \text{ gegeben ist} \end{cases}$$

die Abschätzung

$$\left|\frac{f(a)-\widetilde{f}(\widetilde{a})}{f(a)}\right| \leq \varepsilon_{dat,f} + \tag{6.7}$$
$$+\begin{cases}\dfrac{\Delta(f)}{|f(a)|}, & \text{falls } |f(\widetilde{a})| \in [0, \texttt{MinReal}) \\ \varepsilon(f)(\varepsilon_{dat,f}+1), & \text{falls } |f(\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}]\end{cases}$$

Anmerkungen:

1. Manchmal wird von der Approximation $\widetilde{f} \approx f$ verlangt, dass sie dieselben Nullstellen wie f hat, d.h. $f(\widetilde{x}) = 0 \implies \widetilde{f}(\widetilde{x}) = 0$. Die oben angegebenen Fehlerschranken behalten jedoch ihre Gültigkeit auch dann, wenn diese Eigenschaft nicht erfüllt ist.

2. In [3] wird auf Seite 21 bei der Berechnung des absoluten Rundungsfehlers die Summe $\varepsilon(f)|f(\widetilde{a})| + \Delta(f)$ gebildet. Dadurch wird zwar die in (6.6) erforderliche Fallunterscheidung vermieden, in speziellen Fällen liefert die Addition der Teilfehler $\varepsilon(f)|f(\widetilde{a})|$ und $\Delta(f)$ jedoch eine erhebliche Überschätzung des tatsächlichen Fehlers. Wir benutzen daher die in (6.6) und (6.7) angegebenen Fallunterscheidungen und vermeiden dadurch die genannten Überschätzungen. Vergleichen Sie dazu bitte auch das Beispiel 2 auf Seite 159.

3. Ist jetzt nach (6.6) oder (6.7) der absolute oder relative Fehler für alle exakten Funktionsargumente $a \in I\!R$ eines vorgegebenen Maschinenintervalls $A \in IR$ abzuschätzen, so muss zunächst A so in Teilintervalle zerlegt werden, dass in jedem Teilintervall die Werte $f(\widetilde{a})$ alle entweder nur im normalisierten Bereich oder alle nur im Unterlaufbereich $U = (-\texttt{Minreal}, \texttt{Minreal})$ liegen. Für jedes Teilintervall ist dann eine Oberschranke des gewünschten Fehlers zu berechnen, und bez. ganz A ist danach das Maximum dieser Oberschranken zu bilden.

4. Falls Standardfunktionen, wie z.B. $\cosh(x), \coth(x)$ und $\operatorname{arcosh}(x)$, keine Funktionswerte $f(\widetilde{a})$ im Unterlaufbereich besitzen, so ist die Fallunterscheidung in (6.6) und (6.7) natürlich nicht erforderlich. Aber auch dann, wenn Funktionen, wie z.B. $\sqrt{x}$, nur den einen Funktionswert $\sqrt{0} = \texttt{sqrt}(0) = 0$ im Unterlaufbereich besitzen, kann wegen $\Delta(f) = 0$ auf die Fallunterscheidung ebenfalls verzichtet werden!

5. Gelegentlich kann gezeigt werden, dass die Fehlerschranken ihr Maximum an einem Randpunkt von A annehmen, siehe z.B. die Funktionen: $\sqrt{x},\ e^x,\ \ln(x)$.

Als Beispiel zur Anwendung der Sätze 6.1.1 und 6.1.2 werden im folgenden Abschnitt Fehlerabschätzungen für einige mathematische Standardfunktionen durchgeführt.

6.2 Fehlerschranken für spezielle Standardfunktionen

6.2.1 Die Funktion sqrt$(x) \approx \sqrt{x}$

Mit dem folgenden Satz kann gezeigt werden, dass bei der Quadratwurzel die absolute Fehlerschranke bei gegebenem $\Delta(a)$ ihr Maximum auf dem Intervallrand $A = [\underline{a}, \overline{a}]$ annimmt, vgl. die Herleitung des absoluten Fehlers auf Seite 152.

Satz 6.2.1 Für $c_1, c_2 \geq 0$ nimmt die Funktion $g : X \to I\!R, X = [\underline{x}, \overline{x}] \in I\!I\!R,\ \underline{x} > c_1$, die durch

$$g(x) := \frac{c_1}{2\sqrt{x - c_1}} + c_2 \left(\frac{c_1}{2\sqrt{x - c_1}} + \sqrt{x} \right)$$

definiert sei, ihr Maximum am Rand von X an, d.h.

$$\max_{x \in X} g(x) = \max_{x \in [\underline{x}, \overline{x}]} g(x).$$

Beweis: Im Fall $c_1 \cdot c_2 = 0$ ist $g(x) \equiv 0$ oder g ist monoton. Im Fall $c_1 \cdot c_2 \neq 0$ ist der Beweis geführt, wenn man gezeigt hat, dass die für $x > c_1$ stetig differenzierbare Funktion g genau ein relatives Minimum besitzt. Wegen

$$\lim_{x \to c_1 + 0} g(x) = +\infty \quad \text{und} \quad \lim_{x \to +\infty} g(x) = +\infty$$

besitzt g als stetige Funktion für $x > c_1$ mindestens ein relatives Minimum. Es bleibt also zu zeigen, dass g für $x > c_1$ neben diesem Minimum keinen weiteren Extremwert besitzen kann. Einfache Rechnungen zeigen:

$$g'(x) = 0 \quad \Longleftrightarrow \quad c_1(1 + c_2)\sqrt{x} = 2c_1(x - c_1)^{3/2}$$

Da beide Seiten der rechten Gleichung wegen $x > c_1$ positiv sind, ist das Quadrieren eine Äquivalenzumformung, und man erhält

$$g'(x) = 0 \quad \Longleftrightarrow \quad c_1^2(1 + c_2)^2 \cdot x = 4c_2^2 \cdot (x - c_1)^3$$

Mit $\alpha := 4c_2^2/[c_1^2(1 + c_2)^2] > 0$ und $h(x) := x - \alpha \cdot (x - c_1)^3$ gilt dann

$$g'(x) = 0 \quad \Longleftrightarrow \quad h(x) = 0,$$

wobei zu zeigen ist, dass $h(x)$ für $x \geq c_1$ neben der einen existierenden Nullstelle keine weitere Nullstelle besitzen kann. Es gilt $h(c_1) = c_1 > 0$ und wegen $h''(x) = -6\alpha \cdot (x - c_1) < 0$ ist $h(x)$ für $x > c_1$ stets rechtsgekrümmt. Falls nun $h(x)$ neben der schon nachgewiesenen Nullstelle eine zweite Nullstelle besitzen würde, so müsste $h(x)$ in wenigstens einem Teilbereich zwischen diesen Nullstellen linksgekrümmt sein, was wegen $h''(x) < 0$ nicht sein kann, d.h. $h(x)$ und damit auch $g'(x)$ kann nur eine Nullstelle besitzen. ∎

Wir zeigen jetzt, dass für die Quadratwurzel die Voraussetzungen des Satzes 6.1.1 durch den **IEEE**-754-Standard erfüllt werden. Die Funktion $f(x) = \sqrt{x}$ ist differenzierbar auf dem Intervall

$$\widetilde{A} = A + [-\Delta(a), \Delta(a)] = A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)],$$

wenn $\texttt{Inf}(\widetilde{A}) > 0$ ist. Für die Maschinennäherung $\widetilde{f} \approx f$ verlangt der **IEEE**-754-Standard

$$|f(\widetilde{x}) - \widetilde{f}(\widetilde{x})| \leq \varepsilon(h) \cdot |f(\widetilde{x})|, \qquad \varepsilon(h) = 2^{-52} = 2.220446\ldots \cdot 10^{-16}$$

für alle $\widetilde{x} \in S(2.53) = S$, mit $|f(\widetilde{x})| \in [\texttt{MinReal}, \texttt{MaxReal}]$ und

$$|f(\widetilde{x}) - \widetilde{f}(\widetilde{x})| = 0$$

für alle $\widetilde{x} \in S$, mit $|f(\widetilde{x})| \in [\texttt{0}, \texttt{MinReal})$, wobei die letzte Bedingung nur für $\widetilde{x} = 0$ erfüllt ist und nach **IEEE**-754 $\widetilde{\sqrt{(}0)} = 0$ gilt. Damit sind die Voraussetzungen von Satz 6.1.1 erfüllt, und man erhält mit

$$\Delta_{dat,sqrt} = \left| \frac{\Delta(a)}{2\sqrt{a + [-\Delta(a), \Delta(a)]}} \right| = \frac{\Delta(a)}{2\sqrt{a - \Delta(a)}} = \frac{\varepsilon(a)\sqrt{a}}{2\sqrt{1 - \varepsilon(a)}}$$

für den absoluten Gesamtfehler die Abschätzung

$$\left| \sqrt{a} - \widetilde{\sqrt{\widetilde{a}}} \right| \leq \Delta_{dat,sqrt} + \varepsilon(h) \cdot \left(\Delta_{dat,sqrt} + |\sqrt{a}| \right)$$

Setzt man $\texttt{Inf}(\widetilde{A}) > 0$ voraus, so ist der Radikand $a - \Delta(a)$ stets positiv, und man erhält für den absoluten Fehler die Schranken

$$\begin{aligned}
\left| \sqrt{a} - \widetilde{\sqrt{\widetilde{a}}} \right| &\leq \frac{\Delta(a)}{2\sqrt{a - \Delta(a)}} + \varepsilon(h) \left(\frac{\Delta(a)}{2\sqrt{a - \Delta(a)}} + \sqrt{a} \right) \\
&\overset{\text{Satz 6.2.1}}{\leq} \max_{a \in \{\underline{a}, \overline{a}\}} \left(\frac{\Delta(a)}{2\sqrt{a - \Delta(a)}} + \varepsilon(h) \left(\frac{\Delta(a)}{2\sqrt{a - \Delta(a)}} + \sqrt{a} \right) \right) \\
\left| \sqrt{a} - \widetilde{\sqrt{\widetilde{a}}} \right| &\leq \frac{\varepsilon(a)\sqrt{a}}{2\sqrt{1 - \varepsilon(a)}} + \varepsilon(h) \left(\frac{\varepsilon(a)\sqrt{a}}{2\sqrt{1 - \varepsilon(a)}} + \sqrt{a} \right) \\
&\leq |\sqrt{A}| \left(\frac{\varepsilon(a)}{2\sqrt{1 - \varepsilon(a)}} + \varepsilon(h) \left(\frac{\varepsilon(a)}{2\sqrt{1 - \varepsilon(a)}} + 1 \right) \right)
\end{aligned}$$

Aus diesen Schranken für den absoluten Fehler ergeben sich die Schranken für den relativen Fehler entsprechend zu

$$\begin{aligned}\left|\frac{\sqrt{a}-\widetilde{\sqrt{\widetilde{a}}}}{\sqrt{a}}\right| &\leq \frac{\frac{\Delta(a)}{a}}{2\sqrt{1-\frac{\Delta(a)}{a}}}+\varepsilon(h)\left(\frac{\frac{\Delta(a)}{a}}{2\sqrt{1-\frac{\Delta(a)}{a}}}+1\right)\\ &\leq \frac{\frac{\Delta(a)}{\langle A\rangle}}{2\sqrt{1-\frac{\Delta(a)}{\langle A\rangle}}}+\varepsilon(h)\left(\frac{\frac{\Delta(a)}{\langle A\rangle}}{2\sqrt{1-\frac{\Delta(a)}{\langle A\rangle}}}+1\right)\\ &\leq \frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+\varepsilon(h)\left(\frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+1\right)\end{aligned}$$

Die bisherigen Ergebnisse sind für alle $a \in A = [\underline{a}, \overline{a}]$ in nachfolgender Tabelle noch einmal zusammengefasst. A ist dabei das Intervall der exakten Argumente, und $\Delta(a)$ bzw. $\varepsilon(a)$ sind die absoluten bzw. relativen Fehlerschranken der eventuell fehlerbehafteten Argumente $\widetilde{a} \in \widetilde{A} = A \diamondsuit [-\Delta(a), \Delta(a)]$. Voraussetzung ist $\texttt{Inf}(\widetilde{A}) > 0$.

abzuschätzen	gegeben	**Gesamtfehlerabschätzung**
$\left\lvert\sqrt{a}-\widetilde{\sqrt{\widetilde{a}}}\right\rvert$	$\Delta(a)$	$\max\limits_{a\in\{\underline{a},\overline{a}\}}\left(\frac{\Delta(a)}{2\sqrt{a-\Delta(a)}}+\varepsilon(h)\left(\frac{\Delta(a)}{2\sqrt{a-\Delta(a)}}+\sqrt{a}\right)\right)$
	$\varepsilon(a)$	$\lvert\sqrt{A}\rvert\left(\frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+\varepsilon(h)\left(\frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+1\right)\right)$
$\left\lvert\frac{\sqrt{a}-\widetilde{\sqrt{\widetilde{a}}}}{\sqrt{a}}\right\rvert$	$\Delta(a)$	$\frac{\frac{\Delta(a)}{\langle A\rangle}}{2\sqrt{1-\frac{\Delta(a)}{\langle A\rangle}}}+\varepsilon(h)\left(\frac{\frac{\Delta(a)}{\langle A\rangle}}{2\sqrt{1-\frac{\Delta(a)}{\langle A\rangle}}}+1\right)$
	$\varepsilon(a)$	$\frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+\varepsilon(h)\left(\frac{\varepsilon(a)}{2\sqrt{1-\varepsilon(a)}}+1\right)$

Tabelle 6.1: Absolute und relative Fehler der Quadratwurzel

Die in den vier Zeilen der oberen Tabelle angegebenen Fehlerschranken werden mit den folgenden vier Funktionen

`abs_sqrt_abs()` `abs_sqrt_rel()` `rel_sqrt_abs()` `rel_sqrt_rel()`

berechnet, die vom Modul `abs_relh` exportiert werden.

6.2.1.1 Ein Beispiel

Mit dem nachfolgenden Programm `book_expl21.cpp` wird zu einem vorgegebenen Intervall $A \in IR$ der exakten Argumente und zu einer absoluten Fehlerschranke $\Delta(a) = \texttt{dela} \in S(2,53)$ der fehlerbehafteten Argumente $\widetilde{a} \in \widetilde{A} = A \diamondplus [-\Delta(a), \Delta(a)]$ eine Schranke für den absoluten und relativen Fehler berechnet. Dabei werden also in der Tabelle 6.1 von Seite 153 die Gesamtfehlerschranken aus Zeile 1 und 3 der letzten Spalte ausgewertet.

```
// Programm:  book_expl21.cpp
#include "abs_relh.hpp"   // Wegen abs_sqrt_abs(), rel_sqrt_abs()
#include <iostream>       // Wegen cout;
using namespace cxsc;
using namespace std;

int main()
{
    interval A,R;
    real dela,abs,rel;

    while(1) // Endlosschleife
    {
        cout << "Intervall A = [a1,a2] = ? ";  cin >> A;
        cout << "dela = ? ";  cin >> RndUp >> dela;
        abs_sqrt_abs(A,dela,R,abs); // abs: Fehlerschr. der 1. Zeile
        cout << RndUp << "absoluter Fehler = " << abs << endl;
        rel_sqrt_abs(A,dela,R,rel); // rel: Fehlerschr. der 3. Zeile
        cout << RndUp << "relativer Fehler = " << rel << endl << endl;
    }
}
```

Die Programmergebnisse sind in nachfolgender Tabelle zusammengestellt:

$A \in IR$	$\Delta(a)$	absoluter Fehler	relativer Fehler
$[10^{-4}, 1]$	0	$2.220447 \cdot 10^{-16}$	$2.220447 \cdot 10^{-16}$
$[10^{-4}, 1]$	10^{-17}	$5.022205 \cdot 10^{-16}$	$5.022205 \cdot 10^{-14}$
$[10^{-4}, 1]$	10^{-16}	$5.002221 \cdot 10^{-15}$	$5.002221 \cdot 10^{-13}$
$[4, 16]$	0	$8.881785 \cdot 10^{-16}$	$2.220447 \cdot 10^{-16}$
$[4, 16]$	10^{-17}	$8.894285 \cdot 10^{-16}$	$2.232947 \cdot 10^{-16}$
$[4, 16]$	10^{-16}	$9.006785 \cdot 10^{-16}$	$2.345447 \cdot 10^{-16}$

6.2.2 Die Funktion $\texttt{exp}(x) \approx e^x$

Mit dem folgenden Satz kann gezeigt werden, dass bei der Exponentialfunktion die relative Fehlerschranke bei gegebenem $\varepsilon(a)$ ihr Maximum auf dem Intervallrand von $A = [\underline{a}, \overline{a}] \subset D_{exp} = I\!R$ annimmt.

Satz 6.2.2 Für $c_1, c_2, c_3 \geq 0$ nimmt die Funktion $g : X \to I\!R, X = [\underline{x}, \overline{x}] \in I\!I\!R$, die durch

$$g(x) := c_1|x|\, e^{c_1|x|} + c_2 \left(c_1|x|e^{c_1|x|} + 1\right) + \frac{c_3}{e^x}$$

definiert sei, ihr Maximum am Rand von X an, d.h.

$$\max_{x \in X} g(x) = \max_{x \in [\underline{x}, \overline{x}]} g(x).$$

Beweis: Es gilt

$$g'(x) = (-c_1 + {c_1}^2 x)e^{-c_1 x}(1 + c_2) - \frac{c_3}{e^x} \leq 0 \quad \text{für} \quad x < 0$$

und

$$g''(x) = (2{c_1}^2 + {c_1}^3 x)e^{c_1 x}(1 + c_2) + \frac{c_3}{e^x} \geq 0 \quad \text{für} \quad x > 0,$$

d.h. g fällt monoton für $x < 0$ und ist konvex für $x > 0$. Mit der Stetigkeit von g, insbesondere in $x = 0$, folgt damit die Behauptung ∎

Bevor wir die Schranken für den absoluten bzw. relativen Fehler angeben, leiten wir zunächst eine Schranke für den fortgepflanzten absoluten Datenfehler her. Nach (6.5) von Seite 148 erhält man bei gegebenem $\Delta(a)$

$$\begin{aligned} \Delta_{dat,\exp} &= \Delta(a) \cdot \left| e^{a+[-\Delta(a),\Delta(a)]} \right| = \Delta(a) \cdot e^a \cdot e^{\Delta(a)} \\ &\leq e^{\overline{a}} \cdot \Delta(a) \cdot e^{\Delta(a)} \quad \forall a \in A, \end{aligned}$$

und nach (6.5) folgt bei gegebenem $\varepsilon(a)$

$$\begin{aligned} \Delta_{dat,\exp} &= |a| \cdot \varepsilon(a) \cdot \left| e^{a \cdot [1-\varepsilon(a), 1+\varepsilon(a)]} \right| = |a| \cdot \varepsilon(a) \cdot \left| e^{a \cdot (1+[-\varepsilon(a), \varepsilon(a)])} \right| \\ &= |a| \cdot \varepsilon(a) \cdot e^a \cdot \left| e^{a \cdot [-\varepsilon(a), \varepsilon(a)])} \right| = |a| \cdot \varepsilon(a) \cdot e^a \cdot e^{|a| \cdot \varepsilon(a)} \\ &\leq |A| \cdot \varepsilon(a) \cdot e^{\overline{a}} \cdot e^{|A| \cdot \varepsilon(a)} \quad \forall a \in A. \quad \text{Wir erhalten damit:} \end{aligned}$$

$$\Delta_{dat,\exp} \leq \begin{cases} e^{\overline{a}} \cdot \Delta(a) \cdot e^{\Delta(a)} \quad \forall a \in A, & \text{falls } \Delta(a) \text{ gegeben ist,} \\ |A| \cdot \varepsilon(a) \cdot e^{\overline{a}} \cdot e^{|A| \cdot \varepsilon(a)} \quad \forall a \in A, & \text{falls } \varepsilon(a) \text{ gegeben ist.} \end{cases} \tag{6.8}$$

Im nächsten Schritt leiten wir jetzt eine Schranke für den fortgepflanzten relativen Datenfehler her. Ist $\Delta(a)$ vorgegeben, so gilt nach Satz 6.1.2 von Seite 149

$$\begin{aligned}\varepsilon_{dat,\exp} &= \Delta(a)\cdot\left|\frac{e^{a+[-\Delta(a),\Delta(a)]}}{e^a}\right| = \Delta(a)\cdot\left|e^{[-\Delta(a),\Delta(a)]}\right| \\ &= \Delta(a)\cdot e^{\Delta(a)}, \quad \text{falls } \Delta(a) \text{ gegeben ist.}\end{aligned}$$

Ist $\varepsilon(a)$ vorgegeben, so erhält man entsprechend

$$\begin{aligned}\varepsilon_{dat,\exp} &= |a|\cdot\varepsilon(a)\cdot\left|\frac{e^{a\cdot[1-\varepsilon(a),1+\varepsilon(a)]}}{e^a}\right| = |a|\cdot\varepsilon(a)\cdot\left|e^{a\cdot[-\varepsilon(a),\varepsilon(a)]}\right| \\ &= |a|\cdot\varepsilon(a)\cdot e^{|a|\cdot\varepsilon(a)}, \quad \text{falls } \varepsilon(a) \text{ gegeben ist. Daraus folgt:}\end{aligned}$$

$$\varepsilon_{dat,\exp} = \begin{cases}\Delta(a)\cdot e^{\Delta(a)}, & \text{falls } \Delta(a) \text{ gegeben ist,}\\ |a|\cdot\varepsilon(a)\cdot e^{|a|\cdot\varepsilon(a)}, & \text{falls } \varepsilon(a) \text{ gegeben ist.}\end{cases} \tag{6.9}$$

Mit (6.8) und (6.9) können wir jetzt zusammen mit (6.6) und (6.7) Fehlerschranken für den absoluten oder relativen Fehler der Exponentialfunktion angeben. Um dabei entscheiden zu können, für welche Maschinenzahlen $\widetilde{a} \in \widetilde{A} := A + [-\Delta(a), \Delta(a)]$ die Funktionswerte $e^{\widetilde{a}}$ im normalisierten Bereich oder im Unterlaufbereich liegen, benötigen wir noch die folgenden Informationen:

$$e^{\widetilde{a}} < \texttt{MinReal} = 2^{-1022} \quad\Longleftrightarrow\quad \widetilde{a} < -1022\cdot\ln(2), \text{ d.h. } e^{\widetilde{a}} \in [0, \texttt{MinReal})$$

Es ist $\texttt{q\_min} := -6231120794008786.0/8796093022208.0$ die zu $-1022\cdot\ln(2)$ nächstgrößere Maschinenzahl, und für die Näherungsfunktion $\texttt{exp}(\widetilde{a}) \approx e^{\widetilde{a}}$ gilt:

$$\widetilde{a} < \texttt{q\_min} \quad\Longleftrightarrow\quad \texttt{exp}(\widetilde{a}) \equiv 0 \quad \forall\, \widetilde{a} \in \widetilde{A} \cap S(2,53);$$

Für die Oberschranke $\Delta(\exp)$ des absoluten Fehlers im Unterlaufbereich gilt damit:

$$\widetilde{a} < \texttt{q\_min} \quad\Longrightarrow\quad |e^{\widetilde{a}} - \texttt{exp}(\widetilde{a})| < \Delta(\exp) := \texttt{MinReal} = 2^{-1022}.$$

Für alle $a \in A = [\underline{a}, \overline{a}]$ und $\widetilde{A} = A + [-\Delta(a), \Delta(a)]$ erhält man mit (6.8), (6.6) bei gegebenem $\Delta(a)$:

$$\begin{aligned}|e^a - \texttt{exp}(\widetilde{a})| &\le e^{\overline{a}}\Delta(a)\, e^{\Delta(a)} + \\ &+ \begin{cases}\texttt{MinReal}, & \text{falls } \texttt{Sup}(\widetilde{A}) < \texttt{q\_min}\\ \varepsilon(\exp)\, e^{\overline{a}}\cdot\left(\Delta(a)\, e^{\Delta(a)} + 1\right), & \text{falls } \texttt{Inf}(\widetilde{A}) \ge \texttt{q\_min}\\ \texttt{Max}\left\{\texttt{MinReal}, \varepsilon(\exp)\, e^{\overline{a}}\cdot\left(\Delta(a)\, e^{\Delta(a)} + 1\right)\right\}, & \text{sonst.}\end{cases}\end{aligned} \tag{6.10}$$

Ganz entsprechend erhält man für alle $a \in A = [\underline{a}, \overline{a}]$ und $\widetilde{A} = A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]$ mit (6.8), (6.6) bei gegebener relativer Fehlerschranke $\varepsilon(a)$ der gestörten Funktionsargumente:

$$
\begin{aligned}
&|e^a - \mathtt{exp}(\widetilde{a})| \le |A|\,\varepsilon(a)\,e^{\overline{a}} \cdot e^{|A|\,\varepsilon(a)} + \\
&\quad + \begin{cases} \mathtt{MinReal}, & \text{falls } \mathtt{Sup}(\widetilde{A}) < \mathtt{q\_min} \\ \varepsilon(\exp)\,e^{\overline{a}} \cdot \left(|A|\,\varepsilon(a) \cdot e^{|A|\,\varepsilon(a)} + 1\right), & \text{falls } \mathtt{Inf}(\widetilde{A}) \ge \mathtt{q\_min} \\ \mathtt{Max}\left\{\mathtt{MinReal}, \varepsilon(\exp)\,e^{\overline{a}} \cdot \left(|A|\,\varepsilon(a) \cdot e^{|A|\,\varepsilon(a)} + 1\right)\right\}, & \text{sonst.} \end{cases}
\end{aligned} \tag{6.11}
$$

Ist $\Delta(a)$ gegeben, so erhält man mit (6.9) und (6.7) für den relativen Fehler für alle $a \in A = [\underline{a}, \overline{a}]$ und $\widetilde{A} = A + [-\Delta a, \Delta a]$ die Abschätzung:

$$
\begin{aligned}
&\left|\frac{e^a - \mathtt{exp}(\widetilde{a})}{e^a}\right| \le \Delta(a)\,e^{\Delta(a)} + \\
&\quad + \begin{cases} \dfrac{\mathtt{MinReal}}{e^{\underline{a}}}, & \text{falls } \mathtt{Sup}(\widetilde{A}) < \mathtt{q\_min} \\ \varepsilon(\exp) \cdot \left(\Delta(a)\,e^{\Delta(a)} + 1\right), & \text{falls } \mathtt{Inf}(\widetilde{A}) \ge \mathtt{q\_min} \\ \mathtt{Max}\left\{\dfrac{\mathtt{MinReal}}{e^{\underline{a}}}, \varepsilon(\exp) \cdot \left(\Delta(a)\,e^{\Delta(a)} + 1\right)\right\}, & \text{sonst.} \end{cases}
\end{aligned} \tag{6.12}
$$

Ist $\varepsilon(a)$ gegeben und liegt $e^{\widetilde{a}}$ im normalisierten Bereich, so gilt nach (6.9) und (6.7) für den relativen Fehler

$$
\begin{aligned}
\left|\frac{e^a - \mathtt{exp}(\widetilde{a})}{e^a}\right| &\le |a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + \varepsilon(\exp) \cdot \left(|a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + 1\right) \\
&\le \max_{a \in \{\underline{a}, \overline{a}\}} \left\{|a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + \varepsilon(\exp) \cdot \left(|a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + 1\right)\right\}, \\
&\quad \text{falls } \mathtt{Inf}(\widetilde{A}) \ge \mathtt{q\_min}
\end{aligned}
$$

Ist $\varepsilon(a)$ gegeben und liegt $e^{\widetilde{a}}$ im Unterlaufbereich, so gilt nach (6.9) und (6.7) für den relativen Fehler

$$
\begin{aligned}
\left|\frac{e^a - \mathtt{exp}(\widetilde{a})}{e^a}\right| &\le |a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + \mathtt{MinReal} \cdot e^{-a}, \text{ falls } e^{\widetilde{a}} \in [0, \mathtt{MinReal}) \\
&\le |\underline{a}|\,\varepsilon(a) \cdot e^{|\underline{a}|\,\varepsilon(a)} + \mathtt{MinReal} \cdot e^{|\underline{a}|}, \text{ falls } \mathtt{Sup}(\widetilde{A}) < \mathtt{q\_min},
\end{aligned}
$$

dabei wird ausgenutzt, dass die Funktion $h(a) := |a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)} + \mathtt{MinReal} \cdot e^{-a}$ für $a < 0$ monoton fällt und $a < 0$ wegen $a \le \overline{a} = \mathtt{Sup}(A) \le \mathtt{Sup}(\widetilde{A}) < \mathtt{q\_min} < 0$ erfüllt ist.

Mit den bisherigen Ergebnissen können wir jetzt bei vorgegebenem $\varepsilon(a)$ für alle $a \in A = [\underline{a}, \overline{a}]$ und $\widetilde{A} = A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]$ eine Schranke für den relativen Fehler berechnen. Mit der Funktion $u(a) := |a|\,\varepsilon(a) \cdot e^{|a|\,\varepsilon(a)}$ erhält man:

$$\left|\frac{e^a - \texttt{exp}(\widetilde{a})}{e^a}\right| \leq$$

$$\leq \begin{cases} u(\underline{a}) + \texttt{MinReal} \cdot e^{|\underline{a}|}, & \text{falls } \texttt{Sup}(\widetilde{A}) < \texttt{q\_min} \\ \max\limits_{a \in \{\underline{a}, \overline{a}\}} \left\{u(a) + \varepsilon(\exp) \cdot \big(u(a) + 1\big)\right\}, & \text{falls } \texttt{Inf}(\widetilde{A}) \geq \texttt{q\_min} \\ \texttt{Max}\left\{u(\underline{a}) + \texttt{MinReal} \cdot e^{|\underline{a}|}, \max\limits_{a \in \{\texttt{q\_min}, \overline{a}\}} \left\{u(a) + \varepsilon(\exp)\big(u(a) + 1\big)\right\}\right\}, & \\ \text{sonst.} & \end{cases} \tag{6.13}$$

Die in (6.10), (6.11), (6.12), (6.13) angegebenen Fehlerschranken lassen sich mit den folgenden vier Funktionen

`abs_exp_abs()` `abs_exp_rel()` `rel_exp_abs()` `rel_exp_rel()`

berechnen, die vom Modul `abs_relh` exportiert werden.

6.2.2.1 Beispiele

Beispiel 1 (Funktionswerte nur im Unterlaufbereich) Wir wählen als Intervall der exakten Argumente $A = [-800, -708.5]$ und $\Delta(a) = 10^{-13}$. Dann liegen die gestörten Maschinenargumente $\widetilde{a} = a + \Delta_a$, mit $|\Delta_a| \leq \Delta(a)$, in $\widetilde{A} = A + [-\Delta(a), \Delta(a)]$ und wegen $-708.5 + \Delta(a) < -1022 \cdot \ln(2) = \ln(\texttt{MinReal})$ gilt $e^{\widetilde{a}} < \texttt{MinReal}$, so dass die Funktionswerte $e^{\widetilde{a}}$ alle im Unterlaufbereich $[0, \texttt{MinReal})$ liegen. Das Programm `book_expl22.cpp`

```
#include "abs_relh.hpp"  // Wegen abs_exp_abs()
#include <iostream>      // Wegen cout;
using namespace cxsc;   using namespace std;
int main()
{
    real bnd,dela;  interval A,R;
    while(1) // Endlosschleife
    {
        cout << "Intervall A = ? ";  cin >> A;
        cout << "dela = ? ";  cin >> RndUp >> dela;
        abs_exp_abs(A,dela,R,bnd);
        cout << RndUp << "absolute error =  " << bnd << endl;
    }
}
```

liefert dann mit den entsprechenden Eingabedaten nach (6.10), 1. Zeile, für den absoluten Fehler die Oberschranke

$$|e^a - \texttt{exp}(\widetilde{a})| \leq 2.225074 \cdot 10^{-308} \quad \forall a \in A = [-800, -708.5],\ \Delta(a) = 10^{-13};$$

Vergrößert man $\Delta(a)$, so macht sich in (6.10) der erste Summand $e^{\overline{a}}\Delta(a)\, e^{\Delta(a)}$ zunehmend bemerkbar, und mit $\Delta(a) = 0.5$ erhält man:

$$|e^a - \texttt{exp}(\widetilde{a})| \leq 3.878851 \cdot 10^{-308} \quad \forall a \in A = [-800, -708.5],\ \Delta(a) = 0.5;$$

Beispiel 2 (Funktionswerte in der unmittelbaren Nähe des Unterlaufbereichs) Wir wählen als Intervall der exakten Argumente $A = [-708.3125, -707]$ und $\Delta(a) = 10^{-13}$. Dann liegen die gestörten Maschinenargumente $\widetilde{a} = a + \Delta_a$, mit $|\Delta_a| \leq \Delta(a)$, in $\widetilde{A} = A + [-\Delta(a), \Delta(a)]$ und wegen $\texttt{Inf}(\widetilde{A}) = -708.3125 - \Delta(a) > -1022 \cdot \ln(2) = \ln(\texttt{MinReal}) = -708.3964\ldots$ befinden sich alle Funktionswerte $e^{\widetilde{a}}$ noch im normalisierten Zahlenbereich, aber schon in der unmittelbaren Nähe des Unterlaufbereichs $U = (-\texttt{MinReal}, +\texttt{MinReal})$. Mit den entsprechenden Eingabedaten liefert das Programm von Seite 158 nach (6.10), 2. Zeile, für den absoluten Fehler die Oberschranke

$$|e^a - \texttt{exp}(\widetilde{a})| \leq 9.016699 \cdot 10^{-321} \quad \forall a \in A = [-708.3125, -707],\ \Delta(a) = 10^{-13};$$

Beachten Sie bitte, dass sich in diesem Beispiel der absolute Fehler um $\texttt{MinReal} := 2^{-1022} = 2.22507\ldots \cdot 10^{-308}$ ganz erheblich vergrößern würde, wenn man nach [3] auf Seite 21 bei der Berechnung des Rundungsfehlers die dortige Schranke $\Delta(f)$ des Unterlaufbereichs zusätzlich addieren würde; vergleichen Sie dazu bitte auch die entsprechenden Bemerkungen auf Seite 150.

Beispiel 3 ($e^{\widetilde{x}} \approx \exp(-\widetilde{x} \odot \widetilde{x})$) In diesem Beispiel soll eine Schranke des relativen Fehlers berechnet werden, wenn der Funktionsterm $e^{-\widetilde{x}^2}$ für alle Maschinenzahlen $\widetilde{x} \in X = [20, 26.5]$ durch $\exp(-\widetilde{x} \odot \widetilde{x})$ approximiert wird. Für alle $\widetilde{x} \in X \cap S(2, 53)$ liegen die Funktionswerte $e^{-\widetilde{x}^2}$ im normalisierten Zahlenbereich. Das Programm `book_expl23.cpp`

```
// Programm:  book_expl23.cpp
#include "abs_relh.hpp"
#include <iostream>    // Wegen cout;

using namespace cxsc;
using namespace std;

int main()
{
```

```
    real rel,dela;
    interval X,A,R;

    while(1) // Endlosschleife
    {
        cout << "Intervall X = ? ";  cin >> X;
        abs_mulh1(X,0,X,0,A,dela);
        rel_exp_abs(-A,dela,R,rel);
        cout << RndUp << "relative error =  " << rel << endl << endl;
    }
}
```

liefert mit dem Eingabeintervall $X = [20, 26.5]$ für den relativen Fehler die Oberschranke:

$$\left| \frac{e^{-\widetilde{x}^2} - \exp(-\widetilde{x} \odot \widetilde{x})}{e^{-\widetilde{x}^2}} \right| \leq 1.561667 \cdot 10^{-13} \quad \forall\, \widetilde{x} \in X \cap S(2, 53);$$

Anmerkungen:

1. Beachten Sie bitte, dass durch `abs_mulh1(X,0,X,0,A,dela)` mit `A` zunächst eine Einschließung aller exakten Produkte $\widetilde{x} \cdot \widetilde{x}$ berechnet wird und dass `dela` eine Oberschranke des absoluten Fehlers bei Auswertung dieser Produkte auf der Maschine liefert. Da die Auswertung des Approximationsterms $\exp(-\widetilde{x} \odot \widetilde{x})$ nach Voraussetzung nur für Maschinenzahlen $\widetilde{x} \in X \cap S(2, 53)$ erfolgen soll, wird der obige zweite und vierte Eingangsparameter auf Null gesetzt.

2. Im Vergleich zur relativen Fehlerschranke $\varepsilon(exp) = 2.357963 \cdot 10^{-16}$ der Exponentialfunktion fällt die berechnete Fehlerschranke $1.561667 \cdot 10^{-13}$ viel zu groß aus, so dass die Approximation von $e^{-\widetilde{x}^2}$ durch $\exp(-\widetilde{x} \odot \widetilde{x})$ nicht akzeptabel ist und die Funktion $e^{-\widetilde{x}^2}$ durch einen anderen Algorithmus mit viel kleinerer Fehlerschranke zu approximieren ist. Der Grund für die große Fehlerschranke ist die starke Änderung der Funktionswerte der Exponentialfunktion, wenn bei großen Funktionsargumenten $\widetilde{x}^2 \approx 702.25$ nur kleine Störungen der berechneten Produkte $\widetilde{x}^2 \approx \widetilde{x} \odot \widetilde{x}$ vorliegen. Wählt man daher $X = [0, 1]$, so reduziert sich der relative Fehler zwar auf $4.578409 \cdot 10^{-16}$, ist jedoch im Vergleich zu $\varepsilon(exp) = 2.357963 \cdot 10^{-16}$ immer noch zu groß!

6.2.3 Die Funktion $\texttt{ln}(x) \approx \ln(x)$

Mit dem folgenden Satz kann gezeigt werden, dass bei der ln-Funktion die relative Fehlerschranke bei gegebenem $\Delta(a)$ ihr Maximum auf dem Intervallrand von $A = [\underline{a}, \overline{a}] \subset D_{ln} = I\!R^+$ annimmt.

Satz 6.2.3 Für $c_1, c_2 \geq 0$ nimmt die Funktion $g : X \to I\!R, X = [\underline{x}, \overline{x}] \in I\!I\!R, \underline{x} > c_1$, die durch

$$g(x) := \frac{c_1}{x - c_1} + c_2 \left(\frac{c_1}{x - c_1} + |\ln(x)| \right)$$

definiert sei, ihr Maximum am Rand von X an, d.h.

$$\max_{x \in X} g(x) = \max_{x \in [\underline{x}, \overline{x}]} g(x).$$

Beweis: Da der Beweis für $c_1 \cdot c_2 = 0$ trivial ist, betrachten wir $c_1 \cdot c_2 \neq 0$. Die für $x > c_1 > 0$ stetige Funktion $g(x)$ ist dann überall differenzierbar, außer in $x = 1$, und es gilt:

$$g'(x) = -\frac{c_1(c_2 + 1)}{(x - c_1)^2} - \frac{c_2}{x} < 0 \quad \text{für} \quad c_1 < x < 1,$$

d.h. $g(x)$ ist in $c_1 < x < 1$ streng monoton fallend. Für $1 < x \leq \underline{x}$ erhält man

$$g'(x) = -\frac{c_1(c_2 + 1)}{(x - c_1)^2} + \frac{c_2}{x}$$

$$g'(x) = 0 \quad \Longleftrightarrow \quad x_{1,2} = \frac{3c_1c_2 + c_1 \pm \sqrt{5c_1^2c_2^2 + 6c_1^2c_2 + c_1^2}}{2c2}, \quad \text{wobei } x_2 \text{ wegen}$$

$$x_2 = \frac{3c_1c_2 + c_1 - \sqrt{5c_1^2c_2^2 + 6c_1^2c_2 + c_1^2}}{2c2} \leq \frac{3c_1c_2 + c_1 - \sqrt{(2c_1c_2 + c_1)^2}}{2c2} = \frac{c_1}{2} < c_1$$

nicht in X enthalten ist. Im Falle $x_1 \in (1, \overline{x}]$ ist dann x_1 in diesem Teilintervall die einzige Nullstelle von $g'(x)$, d.h. nur in x_1 kann $g(x)$ einen relativem Extremwert besitzen, der hier wegen $g''(x_1) > 0$ ein Minimum sein muss, woraus die Behauptung unmittelbar folgt. Liegt x_1 nicht in $(1, \overline{x}]$, so ist dort $g(x)$ entweder monoton fallend oder steigend. Wegen der Stetigkeit ist dann $g(x)$ in ganz X entweder streng monoton fallend, oder $g(x)$ besitzt an der nicht differenzierbaren Stelle $x = 1$ als einzigen Extremwert ein lokales Minimum. ∎

Bevor wir die Schranken für den absoluten bzw. relativen Fehler angeben, leiten wir zunächst eine Schranke für den fortgepflanzten absoluten Datenfehler her. Nach (6.5) von Seite 148 erhält man bei gegebenem $\Delta(a)$ und für $a - \Delta(a) > 0$

$$\begin{aligned} \Delta_{dat,\ln} &= \Delta(a) \cdot \left| \frac{1}{a + [-\Delta(a), \Delta(a)]} \right| = \frac{\Delta(a)}{a - \Delta(a)} \\ &\leq \frac{\Delta(a)}{\underline{a} - \Delta(a)} \quad \forall a \in A = [\underline{a}, \overline{a}], \text{ mit } \underline{a} > 0, \end{aligned}$$

und nach (6.5) folgt bei gegebenem $\varepsilon(a)$ für $1-\varepsilon(a)>0$ und $a>0$

$$\Delta_{dat,\ln} = |a|\,\varepsilon(a)\cdot\left|\frac{1}{a\cdot[1-\varepsilon(a),1+\varepsilon(a)]}\right| = \frac{a\cdot\varepsilon(a)}{a\cdot(1-\varepsilon(a))} = \frac{\varepsilon(a)}{1-\varepsilon(a)}$$

Für den absoluten fortgepflanzten Datenfehler $\Delta_{dat,\ln}$ gilt damit die Abschätzung:

$$\Delta_{dat,\ln} = \begin{cases} \dfrac{\Delta(a)}{\underline{a}-\Delta(a)}, & \text{falls } \Delta(a) \text{ gegeben ist,} \\ \dfrac{\varepsilon(a)}{1-\varepsilon(a)}, & \text{falls } \varepsilon(a) \text{ gegeben ist.} \end{cases} \tag{6.14}$$

Die ln-Funktion besitzt nur das eine Maschinenargument $\widetilde{a}=1$, dessen Funktionswert $\ln(1)=\texttt{ln}(1)=0$ im Unterlaufbereich $U=(-\texttt{MinReal},+\texttt{MinReal})$ liegt, so dass $\Delta(\ln)=0$ gesetzt werden kann, denn die Funktionswerte $\ln(\texttt{pred}(1))$ und $\ln(\texttt{succ}(1))$ befinden sich bereits im normalisierten Bereich. Für den absoluten Fehler erhält man dann bei gegebenem $\Delta(a)$ nach (6.6) und (6.14) und mit Satz 6.2.3 die Abschätzung:

$$|\ln(a)-\texttt{ln}(\widetilde{a})| \le \begin{cases} \dfrac{\Delta(a)}{\underline{a}-\Delta(a)}, \text{ falls } \widetilde{A}=A+[-\Delta(a),\Delta(a)]=[1,1] \\ \max\limits_{a\in\{\underline{a},\overline{a}\}}\left\{\dfrac{\Delta(a)}{a-\Delta(a)}+\varepsilon(\ln)\left(\dfrac{\Delta(a)}{a-\Delta(a)}+|\ln(a)|\right)\right\}, \text{ sonst.} \end{cases}$$

Der Fall $\widetilde{A}=[1,1]$ kann jedoch nur eintreten, wenn $A=[1,1]$ und $\Delta(a)=0$, so dass obige Abschätzung weiter vereinfacht werden kann. Bei gegebenem $\Delta(a)$ gilt daher für $\underline{a}-\Delta(a)>0$:

$$\begin{aligned} &|\ln(a)-\texttt{ln}(\widetilde{a})| \le \\ &\le \max_{a\in\{\underline{a},\overline{a}\}}\left\{\frac{\Delta(a)}{a-\Delta(a)}+\varepsilon(\ln)\left(\frac{\Delta(a)}{a-\Delta(a)}+|\ln(a)|\right)\right\}, \quad \forall a\in A=[\underline{a},\overline{a}]. \end{aligned} \tag{6.15}$$

Ganz analog erhält man bei gegebenem $\varepsilon(a)$ nach (6.6) und (6.14) und mit Satz 6.2.3 unter den Voraussetzungen $1-\varepsilon(a)>0$ und $\underline{a}>0$ die Abschätzung:

$$\begin{aligned} &|\ln(a)-\texttt{ln}(\widetilde{a})| \le \\ &\le \frac{\varepsilon(a)}{1-\varepsilon(a)}+\varepsilon(\ln)\cdot\left(\frac{\varepsilon(a)}{1-\varepsilon(a)}+|\ln(A)|\right), \quad \forall a\in A=[\underline{a},\overline{a}]. \end{aligned} \tag{6.16}$$

Bevor wir jetzt den relativen Fehler abschätzen, bestimmen wir zunächst nach Seite 149 Schranken des relativen fortgepflanzten Datenfehlers $\varepsilon_{dat,\ln}$. Bei gegebenem $\Delta(a)$ gilt für $\underline{a}-\Delta(a)>0$

$$\begin{aligned} \varepsilon_{dat,\ln} &= \Delta(a)\cdot\left|\frac{1}{(a+[-\Delta(a),\Delta(a)])\cdot\ln(a)}\right| \\ &\le \frac{\Delta(a)}{(\underline{a}-\Delta(a))\cdot\langle\ln(A)\rangle} \quad \forall a\in A=[\underline{a},\overline{a}],\ \underline{a}>0, \end{aligned}$$

und bei gegebenem $\varepsilon(a)$ erhält man für $1-\varepsilon(a) > 0$:

$$\begin{aligned} \varepsilon_{dat,\ln} &= |a|\,\varepsilon(a) \cdot \left| \frac{1}{(a \cdot [1-\varepsilon(a), 1+\varepsilon(a)]) \cdot \ln(a)} \right| \\ &\leq \frac{\varepsilon(a)}{(1-\varepsilon(a)) \cdot \langle \ln(A) \rangle} \quad \forall a \in A = [\underline{a}, \overline{a}],\ \underline{a} > 0 \end{aligned}$$

Für den relativen fortgepflanzten Datenfehler gilt damit die Abschätzung:

$$\varepsilon_{dat,\ln} = \begin{cases} \dfrac{\Delta(a)}{(\underline{a} - \Delta(a)) \cdot \langle \ln(A) \rangle}, & \text{falls } \Delta(a) \text{ gegeben ist,} \\ \dfrac{\varepsilon(a)}{(1-\varepsilon(a)) \cdot \langle \ln(A) \rangle}, & \text{falls } \varepsilon(a) \text{ gegeben ist.} \end{cases} \tag{6.17}$$

Dabei muss $\underline{a} - \Delta(a) > 0$ bzw. $1 - \varepsilon(a) > 0$ vorausgesetzt werden. Ist zusätzlich $\Delta(a) = 0$ bzw. $\varepsilon(a) = 0$, so definiert man auch im Fall $a = \widetilde{a} = 1 \in A$ den relativen fortgepflanzten Datenfehler zu: $\varepsilon_{dat,\ln} = 0$.

Ist bei gegebenem $\Delta(a)$ der relative Fehler $\left|\frac{\ln(a) - \mathtt{ln}(\widetilde{a})}{\ln(a)}\right|$ für alle $a \in A = [\underline{a}, \overline{a}]$ und damit für alle Maschinenzahlen[1] $\widetilde{a} \in \widetilde{A} = A + [-\Delta(a), \Delta(a)]$ abzuschätzen, so liefert die **C-XSC** Anweisung

```
if ( a - Δ(a) ≤ 0 || Δ(a) != 0 && 1 ∈ Ã ) cerr << ... ;
```

eine Fehlermeldung mit Programmabbruch, andernfalls wird nach (6.17) und (6.7) der relative Fehler abgeschätzt durch[2]:

$$\begin{aligned} &\left|\frac{\ln(a) - \mathtt{ln}(\widetilde{a})}{\ln(a)}\right| \leq \\ &\leq \begin{cases} \varepsilon(\ln),\ \text{falls } \Delta(a) = 0, \\ \dfrac{\Delta(a)}{(\underline{a} - \Delta(a)) \cdot \langle \ln(A) \rangle} + \varepsilon(\ln) \left(\dfrac{\Delta(a)}{(\underline{a} - \Delta(a)) \cdot \langle \ln(A) \rangle} + 1 \right),\ \text{sonst.} \end{cases} \end{aligned} \tag{6.18}$$

Ist bei gegebenem $\varepsilon(a)$ der relative Fehler $\left|\frac{\ln(a) - \mathtt{ln}(\widetilde{a})}{\ln(a)}\right|$ für alle $a \in A = [\underline{a}, \overline{a}]$ und damit für alle Maschinenzahlen[3] $\widetilde{a} \in \widetilde{A} = A \cdot [1-\varepsilon(a), 1+\varepsilon(a)]$ abzuschätzen, so liefert die **C-XSC** Anweisung

```
if ( Inf(Ã) ≤ 0 || ε(a) != 0 && (1 ∈ Ã || 1 - ε(a) ≤ 0) ) cerr << ... ;
```

[1] Es gilt: $\widetilde{a} = a + \Delta_a \in S(2,53)$, mit $|\Delta_a| \leq \Delta(a)\ \forall a \in A$.
[2] Im Falle $\Delta(a) = 0 \wedge A = [1,1]$ wird der relative Fehler wegen $\ln(1) = \mathtt{ln}(1) = 0$ auf Null gesetzt.
[3] Es gilt: $\widetilde{a} = a\,(1+\varepsilon_a) \in S(2,53)$, mit $|\varepsilon_a| \leq \varepsilon(a)\ \forall a \in A$.

eine Fehlermeldung mit Programmabbruch, andernfalls wird nach (6.17) und (6.7) der relative Fehler abgeschätzt durch[4]:

$$\left|\frac{\ln(a) - \mathtt{ln}(\widetilde{a})}{\ln(a)}\right| \leq$$

$$\leq \begin{cases} \varepsilon(\ln), \text{ falls } \Delta(a) = 0, \\ \dfrac{\varepsilon(a)}{(1-\varepsilon(a)) \cdot \langle \ln(A) \rangle} + \varepsilon(\ln)\left(\dfrac{\varepsilon(a)}{(1-\varepsilon(a)) \cdot \langle \ln(A) \rangle} + 1\right), \text{ sonst.} \end{cases} \tag{6.19}$$

Die in (6.15), (6.16), (6.18), (6.19) angegebenen Fehlerschranken lassen sich mit den folgenden vier Funktionen

`abs_ln_abs()` `abs_ln_rel()` `rel_ln_abs()` `rel_ln_rel()`

berechnen, die vom Modul `abs_relh` exportiert werden.

6.2.4 Beispiele

Beispiel 1 (Gegebene absolute Fehlerschranke $\Delta(a)$ der Funktionsargumente) Im folgenden Programm wählt der Anwender das Intervall A der exakten Argumente $a \in A$ und die Fehlerschranke $\Delta(a)$ der gestörten Funktionsargumente $\widetilde{a} = a + \Delta_a$, mit $|\Delta_a| \leq \Delta(a)$. Es gilt dann $\widetilde{a} \in \widetilde{A} = A + [-\Delta(a), \Delta(a)]$, und $\mathtt{Inf}(A) - \Delta(a) > 0$ muss erfüllt sein. Bei der Berechnung des relativen Fehlers darf im Fall $\Delta(a) \neq 0$ $\widetilde{a} = a = 1$ nicht in A enthalten sein. Quelltext des Programms `book_expl24.cpp`:

```
#include "abs_relh.hpp"  // Wegen abs_ln_abs(), rel_ln_abs();
#include <iostream>      // Wegen cout;
using namespace cxsc;  using namespace std;
int main()
{
    real rel,abs,dela;
    interval A,R;
    while(1) // Endlosschleife
    {
        cout << "Intervall A = ? ";  cin >> A;
        cout << "Delta(a) = ? "; cin >> RndUp >> dela;
        abs_ln_abs(A,dela,R,abs);
        cout << RndUp << "Absolute error =  " << abs << endl << endl;
        rel_ln_abs(A,dela,R,rel);
        cout << RndUp << "Relative error =  " << rel << endl << endl;
    }
}
```

[4] Im Falle $\varepsilon(a) = 0 \wedge A = [1,1]$ wird der relative Fehler wegen $\ln(1) = \mathtt{ln}(1) = 0$ auf Null gesetzt.

Das Programm `book_expl24.cpp` berechnet garantierte Schranken des absoluten Fehlers $|\ln(a) - \texttt{ln}(\widetilde{a})|$ und des relativen Fehlers $\left|\frac{\ln(a)-\texttt{ln}(\widetilde{a})}{\ln(a)}\right|$ für alle $a \in A$. Einige Programmergebnisse sind in nachfolgender Tabelle zusammengestellt:

$A \in IR$	$\Delta(a)$	absoluter Fehler	relativer Fehler
$[1,1]$	0	0	0
$[1,1]$	10^{-16}	$1.000001 \cdot 10^{-16}$	existiert nicht
$[0.5,2]$	0	$2.037715 \cdot 10^{-16}$	$2.939801 \cdot 10^{-16}$
$[0.5,2]$	10^{-16}	$4.037715 \cdot 10^{-16}$	existiert nicht
$[0.5,1]$	10^{-16}	$4.037715 \cdot 10^{-16}$	existiert nicht
$[1,2]$	10^{-16}	$2.537715 \cdot 10^{-16}$	existiert nicht
$[1.0001,2]$	10^{-16}	$2.537715 \cdot 10^{-16}$	$1.000244 \cdot 10^{-12}$
$[1.00001,2]$	10^{-16}	$2.537715 \cdot 10^{-16}$	$1.000025 \cdot 10^{-11}$
$[1.00001,2]$	10^{-15}	$9.999930 \cdot 10^{-16}$	$9.999980 \cdot 10^{-11}$
$[10^{200}, 2 \cdot 10^{200}]$	10^{184}	$1.356366 \cdot 10^{-13}$	$2.941972 \cdot 10^{-16}$

In den Spalten 3 und 4 sind garantierte Oberschranken des absoluten bzw. relativen Fehlers angegeben.

Beispiel 2 ($\ln(1+x) \approx \texttt{ln}(1 \oplus x)$) In diesem Beispiel soll gezeigt werden, dass die Approximation $\ln(1+x) \approx \texttt{ln}(1 \oplus x)$ für alle Maschinenzahlen $\widetilde{x} \in X \in IR$ eines vorgegebenen Maschinenintervalls X besonders dann einen zu großen relativen Fehler produziert, wenn X hinreichend dicht am Ursprung liegt. Zu berechnen ist also eine Schranke $\varepsilon(\ln(1 \oplus \widetilde{x}))$ des relativen Fehlers

$$\left|\frac{\ln(1+\widetilde{x}) - \texttt{ln}(1 \oplus \widetilde{x})}{\ln(1+\widetilde{x})}\right| \leq \varepsilon(\ln(1 \oplus \widetilde{x})) \quad \forall\, \widetilde{x} \in X \cap S(2,53); \tag{6.20}$$

Im nachfolgenden Programm wird dazu mit dem Funktionsaufruf

```
abs_1px_delh(X,0,A,dela);
```

zunächst der absolute Fehler $|(1+\widetilde{x}) - (1 \oplus \widetilde{x})|$ für alle $\widetilde{x} \in X$ abgeschätzt durch $\Delta(a) \leq$ `dela`, und die exakten Funktionsargumente $a = 1+\widetilde{x}$ der ln-Funktion werden eingeschlossen durch das Intervall $\texttt{A} = 1 \diamondsuit X$, wobei der Operator $\diamondsuit$ die auf der Maschine auszuführende Intervalladdition symbolisiert. Der 2. Eingangsparameter wurde auf `0` gesetzt, da die $\widetilde{x} \in S(2,53)$ als rundungsfehlerfrei vorausgesetzt werden.

```
// Programm:  book_expl25.cpp
#include "abs_relh.hpp" // Wegen rel_ln_abs()
#include "bnd_arh.hpp"  // Wegen abs_1px_delh()
#include <iostream>     // Wegen cout;
using namespace cxsc;   using namespace std;
int main()
{
    real rel,dela;   interval X,A,R;
    while(1) // Endlosschleife
    {
        cout << "Intervall X = ? ";  cin >> X;
        abs_1px_delh(X,0,A,dela);
        cout << "A = 1 + X = " << A << endl;
        cout << "Delta(a)  = " << dela << endl;
        rel_ln_abs(A,dela,R,rel);
        cout << RndUp << "Relative error =  " << rel << endl << endl;
    }
}
```

Das obige Programm `book_expl25.cpp` berechnet die in (6.20) angegebene Schranke $\varepsilon(\ln(1\oplus\widetilde{x}))$ des relativen Fehlers für alle $\widetilde{x} \in X \cap S(2,52)$. Einige Programmergebnisse sind in nachfolgender Tabelle zusammengestellt:

$X \in IR$	$\mathtt{A} = 1 + X$	$\Delta(a)$	$\varepsilon(\ln(1 \oplus \widetilde{x}))$
$[1, 2]$	$[2.000000, 3.000000]$	$2.220446 \cdot 10^{-16}$	$4.541514 \cdot 10^{-16}$
$[0.0001, 2]$	$[1.000099, 3.000000]$	$2.220447 \cdot 10^{-16}$	$2.220630 \cdot 10^{-12}$
$[0.00001, 2]$	$[1.000009, 3.000000]$	$2.220447 \cdot 10^{-16}$	$2.220465 \cdot 10^{-11}$
$[0.000001, 2]$	$[1.000000, 3.000000]$	$2.220447 \cdot 10^{-16}$	$2.220448 \cdot 10^{-10}$
$[-0.5, -0.001]$	$[0.500000, 0.999001]$	$1.110224 \cdot 10^{-16}$	$2.222276 \cdot 10^{-13}$
$[-0.5, -0.0001]$	$[0.500000, 0.999901]$	$1.110224 \cdot 10^{-16}$	$2.220630 \cdot 10^{-12}$

Man erkennt deutlich das starke Anwachsen des relativen Fehlers, wenn das Intervall X immer dichter an den Ursprung rückt, denn die selbst schon fehlerbehaftete `ln`-Funktion wird dann wegen $\Delta(a) > 0$ mit fehlerbehafteten Argumenten ganz in der Nähe ihrer Nullstelle ausgewertet, wodurch sich der relative Fehler stark vergrößert. Die Approximation von $\ln(1 + \widetilde{x})$ kann mit Hilfe der in **C-XSC** implementierten `lnp1`-Funktion sehr viel genauer erfolgen, da deren Algorithmus einen vergleichsweise nur kleinen relativen Höchstfehler von $\varepsilon(\mathtt{lnp1}) = 2.5082 \cdot 10^{-16}$ garantiert.

6.2.5 Die Funktion lnp1$(x) \approx \ln(1+x)$

Die Funktion $f(x) = \ln(1+x)$ wird in der **C-XSC** Umgebung für alle Maschinenzahlen $\widetilde{x} > -1$ approximiert durch $\texttt{lnp1}(\widetilde{x}) \approx \ln(1+\widetilde{x})$, und für die Fehlerschranken gilt folgendes:

$$-\texttt{MinReal} < \widetilde{x} \leq +\texttt{MinReal} \quad \Longrightarrow \quad \ln(1+\widetilde{x}) \in (-\texttt{MinReal}, +\texttt{MinReal}) \quad \text{und}$$
$$|\texttt{lnp1}(\widetilde{x}) - \ln(1+\widetilde{x})| \leq \Delta(f) := \texttt{minreal} = 2^{-1074}; \tag{6.21}$$

$$-1 < \widetilde{x} \leq -\texttt{MinReal} \vee \widetilde{x} > +\texttt{MinReal} \quad \Longrightarrow \quad |\ln(1+\widetilde{x})| \geq \texttt{MinReal} \quad \text{und}$$
$$\left| \frac{\texttt{lnp1}(\widetilde{x}) - \ln(1+\widetilde{x})}{\ln(1+\widetilde{x})} \right| \leq \varepsilon(f) := 2.5082 \cdot 10^{-16}; \tag{6.22}$$

Mit (6.21) und (6.22) sind damit die Voraussetzungen der Sätze 6.1.1 und 6.1.2 von Seite 148 bzw. 149 erfüllt, mit denen wir den absoluten oder relativen Fehler abschätzen können, wenn der exakte Funktionswert $\ln(1+a)$ durch den Maschinenwert $\texttt{lnp1}(\widetilde{a})$ approximiert wird. In den folgenden Abschnitten bezeichnen wir mit A das Maschinenintervall der exakten Funktionsargumente $a > -1$. Wir nehmen weiter an, dass die reellen $a \in A$ auf der Maschine fehlerhaft als $\widetilde{a} \in S(2,53)$ berechnet wurden und dass die folgenden Beziehungen gelten:

$$\widetilde{a} = a + \Delta_a, \qquad \text{mit } |\Delta_a| \leq \Delta(a) \quad \forall a \in A \tag{6.23}$$
$$\widetilde{a} = a \cdot (1+\varepsilon_a), \quad \text{mit } |\varepsilon_a| \leq \varepsilon(a) \quad \forall a \in A \tag{6.24}$$

Die Maschinenzahlen $\widetilde{a}$ werden dann eingeschlossen durch die Maschinenintervalle $\widetilde{A}$, die folgende Bedingung erfüllen müssen: $\widetilde{A} \subset D_f = \{x \,|\, x \in I\!R \wedge x > -1\}$

$$\widetilde{A} = A \diamondsuit [-\Delta(a), +\Delta(a)] \in IR, \qquad \text{falls } \Delta(a) \text{ gegeben ist} \tag{6.25}$$
$$\widetilde{A} = A \diamondsuit [1-\varepsilon(a), 1+\varepsilon(a)] \in IR, \quad \text{falls } \varepsilon(a) \text{ gegeben ist.} \tag{6.26}$$

$\diamondsuit$, $\diamondsuit$ symbolisieren dabei die auf der Maschine auszuführende Intervall-Addition bzw. -Multiplikation.

Abschätzung des absoluten Fehlers bei gegebenem $\Delta(a)$

Für alle exakten $a \in A$ suchen wir jetzt bei gegebenem $\Delta(a)$ eine Oberschranke des absoluten Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$. Nach (6.6) benötigt man dazu die Schranke $\Delta_{dat,f}$ des fortgepflanzten Datenfehlers, die sich nach (6.5) von Seite 148 wie folgt berechnet:

$$\begin{aligned} \Delta_{dat,f} &= \Delta(a) \cdot \left| f'(a + [-\Delta(a), +\Delta(a)]) \right| \\ &= \Delta(a) \cdot \left| \frac{1}{1 + a + [-\Delta(a), +\Delta(a)]} \right| = \frac{\Delta(a)}{1 + a - \Delta(a)} \qquad (6.27) \\ &\leq \frac{\Delta(a)}{1 + \underline{a} - \Delta(a)} \quad \forall a \in A = [\underline{a}, \overline{a}], \quad \text{mit } \; 1 + \underline{a} - \Delta(a) > 0. \qquad (6.28) \end{aligned}$$

Nach (6.6) von Seite 148 ist eine Fallunterscheidung nötig bez. der Lage der exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten bzw. denormalisierten Bereich. Es gilt:

$$-\texttt{MinReal} < \widetilde{a} \le +\texttt{MinReal} \quad\Longrightarrow\quad |\ln(1+\widetilde{a})| \in [0, +\texttt{MinReal}) \tag{6.29}$$

$$-1 < \widetilde{a} \le -\texttt{MinReal} \vee \widetilde{a} > +\texttt{MinReal} \quad\Longrightarrow\quad |\ln(1+\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \tag{6.30}$$

Bei gegebenem $A = [\underline{a}, \overline{a}]$ und $\Delta(a)$, mit $1 + \underline{a} - \Delta(a) > 0$, kann jetzt im Falle $-\texttt{MinReal} < \widetilde{a} \le +\texttt{MinReal}$ für den absoluten Fehler $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ eine Oberschranke wie folgt bestimmt werden. Zunächst berechnet man mit dem Maschinenintervall $D = \widetilde{A} \cap (-\texttt{MinReal}, +\texttt{MinReal}]$ den Durchschnitt von $\widetilde{A}$ und $(-\texttt{MinReal}, +\texttt{MinReal}]$. Zu $\widetilde{a} = \texttt{Inf}(D)$ erhält man dann mit $\texttt{Inf}(D) - \Delta(a)$ den kleinstmöglichen Wert für das zugehörige exakte Argument a, und mit diesem a liefert dann (6.6) von Seite 148 die gesuchte Oberschranke

$$|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)| \le \frac{\Delta(a)}{1+a-\Delta(a)} + 2^{-1074} \quad \forall\, \widetilde{a} \in D. \tag{6.31}$$

Beachten Sie bitte, dass die Rechenoperationen rechts in (6.31) auf der Maschine mit geeigneten Rundungen durchzuführen sind, um die Berechnung einer garantierten Oberschranke zu gewährleisten. Weitere Einzelheiten dazu entnehmen Sie bitte dem Quelltext auf Seite 169.

Liegt $\widetilde{a}$ im Bereich von (6.30), so liegen die exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten Bereich und der absolute Fehler $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ wird nach (6.6) von Seite 148 abgeschätzt durch

$$|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)| \le g_1(a) := \Delta_{dat,f} + \varepsilon(f) \cdot (\Delta_{dat,f} + |\ln(1+a)|),$$

wobei $\Delta_{dat,f}$ nach (6.27) definiert ist. Die Funktion $g_1(a)$ ist dabei von der Form

$$g_1(a) := \frac{c_1}{1+a-c_1} + c_2 \left(\frac{c_1}{1+a-c_1} + |\ln(1+a)| \right)$$

mit $c_1, c_2 \ge 0$ und $1 + a - c_1 > 0$. Nach Seite 161 erhält man $g_1(a)$ aus der dortigen Funktion $g(x)$ durch die lineare Transformation $x = 1 + a$, so dass auch $g_1(a)$ für alle $a \in A = [\underline{a}, \overline{a}]$ ihr Maximum auf dem Rand von A annimmt, d.h. es gilt:

$$\max_{a \in A} g_1(a) = \max_{a \in [\underline{a}, \overline{a}]} g_1(a), \quad \text{falls } 1 + \underline{a} - c_1 > 0.$$

Im Quelltext von Seite 169 erfolgt die Berechnung des obigen Maximums mit Hilfe der Funktion `real lnp1_abs4Max(const real& a, const real& dela)`

```
// Programm:  book_expl26.cpp
#include <iostream>   // For cout, cin;
#include <imath.hpp>  // For lnp1(A)
using namespace cxsc;
using namespace std;

static const real eps_lnp1 =
  5087233976787912.0 / 2028240960365167042394725128601 6.0;
  // relative error bound of the function ln(1+x); 2.5082E-016;

static const interval MR_l = interval(succ(-MinReal),MinReal);

static real lnp1_abs4Max(const real& a, const real& dela)
    // For calculating the maximum; used by abs_lnp1_abs()
{
    interval z;
    real r,r1;
    r1 = addd(1,a);
    r = subd(r1,dela);
    r = divu(dela,r);
    z = abs( lnp1(interval(a)) );
    r1 = addu(Sup(z),r);
    r1 = mulu(r1,eps_lnp1);
    r1 = addu(r1,r);
    return r1;
}

void abs_lnp1_abs(const interval& A, const real& dela,
                  interval& R, real& abs)
{
    interval At,D,z=interval(-dela,dela);
    At = A+z;  D = At+z;  // At = A_tilde
    if (Inf(D)<=-1)
    {
        cerr << "abs_lnp1_abs(): Absolute error doesn't exist!"
             << endl;   exit(1);
    }
    real r,r1;
    if (Disjoint(At,MR_l)) // intersection is empty:
```

```
    {
        abs = lnp1_abs4Max(Inf(A),dela);
        r   = lnp1_abs4Max(Sup(A),dela);
        if (r>abs) abs = r;
    }
    else // intersection not empty:
    {
    // Calculating an absolute upper bound abs only for the exact
    // function values ln(1+wa) lying in the denormalized range:
        D = At & MR_1;  // D: not empty intersection of At and MR_1;
        r = subd(Inf(D),2*dela);
        r = addd(1,r);  // r>0 is guaranteed by: if (Inf(D)<=-1) { }
        r = divu(dela,r);
        abs = addu(r,minreal);  // abs: the desired error bound

        if (Inf(At) <= -MinReal)
        {
            r  = lnp1_abs4Max(Inf(A),dela);
            r1 = addu(-MinReal,dela);
            r1 = lnp1_abs4Max(r1,dela);
            if (r>r1) r1 = r;
            if (r1>abs) abs = r1;
        }

        if (Sup(At)>MinReal)
        {
            r  = lnp1_abs4Max(Sup(A),dela);
            r1 = subd(succ(MinReal),dela);
            r1 = lnp1_abs4Max(r1,dela);
            if (r>r1) r1 = r;
            if (r1>abs) abs = r1;
        }
    }
    R = lnp1(A); // R: inclusion of ln(1+A);
}
int main()
{
    real abs,dela;
    interval A,R;

    while(1) // infinite loop
```

```
    {
        cout << "Intervall A = ? ";  cin >> A;
        cout << "Delta(a)  = ? "; cin >> RndUp >> dela;
        abs_lnp1_abs(A,dela,R,abs);
        cout << "Absolute error = " << RndUp << abs << endl << endl;
    }
}
```

Das obige Programm `book_expl26.cpp` liefert zum gegebenen Maschinenintervall $\texttt{A} \in IR$ der exakten Funktionsargumente $a \in \texttt{A}$ und zu gegebenem $\Delta(a) = \texttt{dela}$ eine garantierte Oberschranke `abs` des absoluten Fehlers:

$$|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)| \leq \texttt{abs} \quad \forall a \in \texttt{A}.$$

Die obige Funktion `abs_lnp1_abs()` wird vom **C-XSC** Modul `abs_relh` exportiert, so dass man das obige Programm auf die Funktion `int main(){}` reduzieren kann, wenn man im Programmkopf zusätzlich `#include "abs_relh.hpp"` einfügt. In der folgenden Tabelle sind zu gegebenem Maschinenintervall `A` und zur gegebenen absoluten Fehlerschranke $\Delta(a) \leq \texttt{dela} \in S(2,53)$ die mit dem obigen Programm berechneten Oberschranken `abs` des absoluten Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ angegeben:

$\texttt{A} \in IR$	$\Delta(a) \leq \texttt{dela}$	abs
$[1,2]$	$1 \cdot 10^{-14}$	$5.173856 \cdot 10^{-15}$
$[1,2]$	$1 \cdot 10^{-16}$	$3.088873 \cdot 10^{-16}$
$[1,2]$	0.0	$2.755540 \cdot 10^{-16}$
$[2,30]$	$1 \cdot 10^{-15}$	$8.935708 \cdot 10^{-16}$
$[30,30]$	$1 \cdot 10^{-15}$	$8.935708 \cdot 10^{-16}$
$[30,30]$	0.0	$8.613127 \cdot 10^{-16}$
$[-1 \cdot 10^{-60}, +1 \cdot 10^{-60}]$	$1 \cdot 10^{-76}$	$3.508201 \cdot 10^{-76}$
$[-1 \cdot 10^{-60}, +1 \cdot 10^{-60}]$	0.0	$2.508201 \cdot 10^{-76}$

In der obigen Tabelle sind die Dezimalzahlen der zweiten Spalte die Eingabewerte für die absolute Fehlerschranke $\Delta(a)$ der Funktionsargumente, wobei diese Dezimalzahlen mit der Anweisung

```
cout << "Delta(a)  = ? "; cin >> RndUp >> dela;
```

zur nächstgrößeren Rasterzahl $\texttt{dela} \in S(2,53)$ gerundet werden.

Abschätzung des absoluten Fehlers bei gegebenem $\varepsilon(a)$

Für alle exakten $a \in A$ suchen wir jetzt bei gegebenem $\varepsilon(a)$ eine Oberschranke des absoluten Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$. Nach (6.6) benötigt man dazu die Schranke $\Delta_{dat,f}$ des fortgepflanzten Datenfehlers, die sich nach (6.5) von Seite 148 wie folgt berechnet:

$$\begin{aligned} \Delta_{dat,f} &= |a|\varepsilon(a) \cdot f'(a \cdot [1-\varepsilon(a), 1+\varepsilon(a)]) \\ &= \begin{cases} h_1(a) := \dfrac{a \cdot \varepsilon(a)}{1 + a(1-\varepsilon(a))} & \text{falls } a \geq 0 \text{ und } \varepsilon(a) \leq 1 \\ h_2(a) := \dfrac{-a \cdot \varepsilon(a)}{1 + a(1+\varepsilon(a))} & \text{falls } a < 0 \text{ und } a(1+\varepsilon(a)) > -1 \end{cases} \end{aligned} \tag{6.32}$$

Da $h_1(a)$ und $h_2(a)$ monoton wachsen bzw. fallen, nimmt $\Delta_{dat,f}$ für alle $a \in A = [\underline{a}, \overline{a}]$ sein Maximum auf dem Intervallrand von A an, d.h. es gilt:

$$\max_{a \in A} \Delta_{dat,f}(a) = \max_{a \in [\underline{a}, \overline{a}]} \Delta_{dat,f}(a), \quad \text{falls } \varepsilon(a) \leq 1 \text{ und } 1 + \underline{a} \cdot (1+\varepsilon(a)) > 0$$

Nach (6.6) von Seite 148 ist jetzt wieder eine Fallunterscheidung nötig bezüglich der Lage der exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten bzw. denormalisierten Bereich. Es gilt:

$$-\texttt{MinReal} < \widetilde{a} \leq +\texttt{MinReal} \implies |\ln(1+\widetilde{a})| \in [0, +\texttt{MinReal}) \tag{6.33}$$

$$\begin{aligned} -1 < \widetilde{a} \leq -\texttt{MinReal} \vee \widetilde{a} > +\texttt{MinReal} &\implies \\ |\ln(1+\widetilde{a})| &\in [\texttt{MinReal}, \texttt{MaxReal}] \end{aligned} \tag{6.34}$$

Bei gegebenem $A = [\underline{a}, \overline{a}]$ und $\varepsilon(a)$, mit $1 + \underline{a} \cdot (1+\varepsilon(a)) > 0$, kann jetzt im Falle $-\texttt{MinReal} < \widetilde{a} \leq +\texttt{MinReal}$ für den absoluten Fehler $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ eine Oberschranke wie folgt bestimmt werden. Zunächst berechnet man mit dem Maschinenintervall $D = \widetilde{A} \cap (-\texttt{MinReal}, +\texttt{MinReal}]$ den Durchschnitt von $\widetilde{A}$ und $(-\texttt{MinReal}, +\texttt{MinReal}]$. Zum kleinsten Maschinenwert $\widetilde{a} = \texttt{Inf}(D)$ erhält man dann mit den aus $\widetilde{a} = a \cdot (1+\varepsilon_a)$ und $0 \leq |\varepsilon_a| \leq \varepsilon(a) < 1$ folgenden Darstellungen

$$\begin{aligned} a &\in \left[\frac{\widetilde{a}}{1+\varepsilon(a)}, \frac{\widetilde{a}}{1-\varepsilon(a)} \right], \quad \text{falls } \widetilde{a} \geq 0 \\ a &\in \left[\frac{\widetilde{a}}{1-\varepsilon(a)}, \frac{\widetilde{a}}{1+\varepsilon(a)} \right], \quad \text{falls } \widetilde{a} < 0 \end{aligned}$$

und den Anweisungen

```
r1 = (sign(r)<0) ? divd(r,one_meps) : divd(r,one_peps);
r1 = Delta_4Max(r1,epsa,one_meps,one_peps);
```

mit `r1` $\in S(2,53)$ eine Oberschranke für den Funktionswert $\Delta_{dat,f}(a)$, wobei mit `r1` aus der ersten der beiden letzten Anweisungen das kleinstmögliche Funktionsargument a berechnet wurde. Ganz entsprechend findet man zum größten Maschinenwert $\widetilde{a} = \texttt{Sup}(D)$ und den Anweisungen

```
r = Sup(D);
r = (sign(r)<0) ? divd(r,one_peps) : divd(r,one_meps);
r = Delta_4Max(r,epsa,one_meps,one_peps);
```

mit dem letzten `r` eine Oberschranke für den Funktionswert $\Delta_{dat,f}(a)$, wobei mit `r` aus der zweiten der drei obigen Anweisung das größtmögliche Funktionsargument a berechnet wurde. Da $\Delta_{dat,f}(a)$ das Maximum auf dem Rand eines vorgegebenen Intervalls $A = [\underline{a}, \overline{a}]$ annimmt, ist daher das Maximum der Werte `r1` und `r` das Maximum des fortgepflanzten Datenfehlers für alle $\widetilde{a} \in D$. Addiert man zu diesem Maximum noch `minreal` $= 2^{-1074}$, so erhält man nach (6.6) von Seite 148 eine Oberschranke für den absoluten Fehler $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ für alle $\widetilde{a} \in D$. Weitere Einzelheiten findet man im Quelltext der Funktion `abs_lnp1_rel()` auf Seite 173.

Liegt $\widetilde{a}$ im Bereich von (6.34), so liegen die exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten Bereich und der absolute Fehler $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ wird nach (6.6) von Seite 148 abgeschätzt durch

$$|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)| \le h(a) := \Delta_{dat,f} + \varepsilon(f) \cdot (\Delta_{dat,f} + |\ln(1+a)|),$$

wobei $\Delta_{dat,f} = \Delta_{dat,f}(a)$ nach (6.32) definiert ist. Da $h_1(a)$ und $|\ln(1+a)|)$ für $a \ge 0$ monoton wachsen und $h_2(a)$ und $|\ln(1+a)|)$ für $a < 0$ monoton fallen, gilt dies auch für die Funktion $h(a)$, die deshalb ihr Maximum über einem vorgegebenen Intervall $A = [\underline{a}, \overline{a}]$ auf dem Intervallrand annimmt, d.h. es gilt:

$$\max_{a \in A} h(a) = \max_{a \in [\underline{a}, \overline{a}]} h(a), \quad \text{falls} \;\; \varepsilon(a) \le 1 \;\; \text{und} \;\; 1 + \underline{a} \cdot (1 + \varepsilon(a)) > 0$$

In den Fällen $\texttt{Inf}(A) \ge 0$ oder $\texttt{Sup}(A) < 0$ kann die Berechnung des Maximums auf nur jeweils eine Auswertung der Funktion $h(a)$ reduziert werden, wobei $h(a)$ mit Hilfe der Funktion `abslnp1_4Max()` ausgewertet wird. Genauere Einzelheiten findet man im nachfolgenden Quelltext der Funktion `abs_lnp1_rel()`, die vom Modul `abs_relh` exportiert wird.

```
void abs_lnp1_rel(const interval& A, const real& epsa,
                  interval& R, real& abs)
// abs is an upper bound of the absolute error |lnp1(wa)-ln(1+a)| for
// all exact function arguments a in A. epsa is a relative error bound
// of the erroneous function arguments wa; |wa-a|<=|a|*epsa for all a
// in A. R is an inclusion of all function values ln(1+a) with a in A.
```

```
{
    real one_peps = addu(1,epsa), one_meps = subd(1,epsa);
    interval At = A * interval(one_meps,one_peps); // At: A_tilde
    if (epsa>=1 || Inf(At)<=-1)
    {
        cerr << "abs_lnp1_rel(): Absolute error doesn't exist!"
             << endl;   exit(1);
    }
    real r,r1,IA=Inf(A),SA=Sup(A);
    if (Disjoint(At,MR_l)) // empty intersection:
    {
        if (IA>=0)
            abs = abslnp1_4Max(SA,epsa,one_meps,one_peps);
        else if (SA<=0)
            abs = abslnp1_4Max(IA,epsa,one_meps,one_peps);
        else
        {
            abs = abslnp1_4Max(IA,epsa,one_meps,one_peps);
            r   = abslnp1_4Max(SA,epsa,one_meps,one_peps);
            if (r>abs) abs = r;
        }
    }
    else // no empty intersection:
    {
        interval D = At & MR_l; // D: not empty intersection
                                // of At and MR_l;
        r = Inf(D);
        r1 = (sign(r)<0) ? divd(r,one_meps) : divd(r,one_peps);
        r1 = Delta_4Max(r1,epsa,one_meps,one_peps); // left value
        r = Sup(D);
        r = (sign(r)<0) ? divd(r,one_peps) : divd(r,one_meps);
        r = Delta_4Max(r,epsa,one_meps,one_peps); // right value
        if (r>r1) r1 = r;  // r1: maximum value
        abs = addu(r1,minreal); // upper bound of the absolute error
                                // for all machine values of D
        if (Inf(At) <= -MinReal)
        {
            r = abslnp1_4Max(IA,epsa,one_meps,one_peps);
            if (r>abs) abs = r;
        }
```

```
        if (Sup(At)>MinReal)
        {
            r = abslnp1_4Max(SA,epsa,one_meps,one_peps);
            if (r>abs) abs = r;
        }
    }
    R = lnp1(A);
} \\ abs_lnp1_rel
```

In der folgenden Tabelle sind zu gegebenem Maschinenintervall `A` und zur gegebenen relativen Fehlerschranke $\varepsilon(a) \leq$ `epsa` $\in S(2,53)$ die mit der obigen Funktion berechneten Oberschranken `abs` des absoluten Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)|$ angegeben:

`A` $\in IR$	$\varepsilon(a) \leq$ `epsa`	`abs`
$[1,2]$	$1 \cdot 10^{-14}$	$6.942221 \cdot 10^{-15}$
$[1,2]$	$1 \cdot 10^{-16}$	$3.422207 \cdot 10^{-16}$
$[1,2]$	0.0	$2.755540 \cdot 10^{-16}$
$[2,30]$	$1 \cdot 10^{-15}$	$1.829055 \cdot 10^{-15}$
$[30,30]$	$1 \cdot 10^{-15}$	$1.829055 \cdot 10^{-15}$
$[30,30]$	0.0	$8.613127 \cdot 10^{-16}$
$[-1 \cdot 10^{-60}, +1 \cdot 10^{-60}]$	$1 \cdot 10^{-16}$	$3.508201 \cdot 10^{-76}$
$[-1 \cdot 10^{-60}, +1 \cdot 10^{-60}]$	0.0	$2.508201 \cdot 10^{-76}$

Abschätzung des relativen Fehlers bei gegebenem $\Delta(a)$

Für alle exakten Funktionsargumente $a \in A$ suchen wir jetzt bei gegebenem $\Delta(a)$ eine Oberschranke des relativen Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)/\ln(1+a)|$. Nach (6.7) benötigt man dazu die Schranke $\varepsilon_{dat,f}$ des fortgepflanzten Datenfehlers, die sich nach Seite 149 unter der Voraussetzung $1 + a - \Delta(a) > 0$ wie folgt berechnet:

$$
\begin{aligned}
\varepsilon_{dat,f} &= \Delta(a) \cdot \left| \frac{f'(a + [-\Delta(a), +\Delta(a)])}{f(a)} \right| \\
&= \left| \frac{\Delta(a)}{(1 + a + [-\Delta(a), +\Delta(a)]) \cdot \ln(1+a)} \right| \\
&= \begin{cases} h_1(a) := \dfrac{\Delta(a)}{(1 + a - \Delta(a)) \cdot \ln(1+a)} & \text{falls } a > 0 \\ h_2(a) := \dfrac{-\Delta(a)}{(1 + a - \Delta(a)) \cdot \ln(1+a)} & \text{falls } a < 0 \end{cases} \qquad (6.35)
\end{aligned}
$$

dabei ist $h_1(a)$ monoton fallend, und die Funktion $h_2(a)$ besitzt im Falle $\Delta(a) > 0$ in ihrem Definitionsbereich $-1 + \Delta(a) < a < 0$ genau ein relatives Minimum. In jedem Intervall $A = [\underline{a}, \overline{a}]$, mit $0 \notin A$ und $1 + \underline{a} - \Delta(a) > 0$, nimmt daher $\varepsilon_{dat,f}(a)$ ihr Maximum auf dem Rand von A an, d.h. es gilt:

$$\max_{a \in A} \varepsilon_{dat,f}(a) = \max_{a \in [\underline{a}, \overline{a}]} \varepsilon_{dat,f}(a), \quad \text{falls } \; 0 \notin A \;\text{ und }\; 1 + \underline{a} - \Delta(a) > 0\,; \tag{6.36}$$

Der Beweis der obigen Aussagen ist trivial und kann dem Leser überlassen werden.

Nach (6.7) von Seite 150 ist jetzt wieder eine Fallunterscheidung nötig bez. der Lage der exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten bzw. denormalisierten Bereich. Es gilt:

$$-\texttt{MinReal} < \widetilde{a} \leq +\texttt{MinReal} \quad \Longrightarrow \quad |\ln(1+\widetilde{a})| \in [0, +\texttt{MinReal}) \tag{6.37}$$

$$\begin{aligned} -1 < \widetilde{a} \leq -\texttt{MinReal} \vee \widetilde{a} > +\texttt{MinReal} \quad &\Longrightarrow \\ |\ln(1+\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] & \end{aligned} \tag{6.38}$$

Nach (6.37) betrachten wir zunächst den Bereich der Maschinenwerte $\widetilde{a}$, für den die exakten Funktionswerte $\ln(1+\widetilde{a})$ im denormalisierten Zahlenbereich liegen. Nach (6.7) ist dann eine Oberschranke des relativen Fehlers gegeben durch:

$$\left| \frac{\texttt{lnp1}(\widetilde{a}) - \ln(1+a)}{\ln(1+a)} \right| \leq g(a) := \varepsilon_{dat,f} + \frac{\texttt{minreal}}{|\ln(1+a)|}, \tag{6.39}$$

wobei $g(a)$ unter der Voraussetzung $1 + a - \Delta(a) > 0$ definiert ist durch:

$$g(a) = \begin{cases} g_1(a) := \dfrac{\Delta(a)}{(1+a-\Delta(a)) \cdot \ln(1+a)} + \dfrac{\texttt{minreal}}{\ln(1+a)} & \text{falls } a > 0 \\ g_2(a) := \dfrac{-\Delta(a)}{(1+a-\Delta(a)) \cdot \ln(1+a)} - \dfrac{\texttt{minreal}}{\ln(1+a)} & \text{falls } a < 0 \end{cases}$$

Da in $g_1(a)$ beide Summanden monoton fallend sind, gilt dies auch für $g_1(a)$ selbst, und eine Oberschranke von $g_1(a)$ erhält man in einem vorgegebenen Intervall $A = [\underline{a}, \overline{a}]$, wenn man $g_1(\underline{a})$ berechnet.

Für $\Delta(a) = 0$ ist $g_2(a)$ monoton wachsend, und für $\Delta(a) > 0$ besitzt $g_2(a)$ in einem vorgegebenen Intervall $A = [\underline{a}, \overline{a}]$ genau ein relatives Minimum, so dass $g_2(a)$ sein Maximum auf dem Rand von A annimmt. Zum Nachweis des Minimums schreiben wir $1 + a = x$ und $\Delta(a) = \Delta$, so dass gezeigt werden muss, dass

$$u(x) := \frac{-\Delta}{(x-\Delta) \cdot \ln(x)} - \frac{\texttt{minreal}}{\ln(x)}$$

in $0 < \Delta < x < 1$ genau ein relatives Minimum besitzt. Elementare Rechnungen zeigen, dass $u'(x) = 0$ äquivalent ist zu

$$\ln(x) + 1 = \frac{\Delta}{x} - \frac{\texttt{minreal}}{\Delta} \cdot \frac{(x-\Delta)^2}{x} =: r(x), \tag{6.40}$$

wobei $\ln(x)+1$ monoton wächst, mit $\ln(x)+1 \le 1$ für $0 < \Delta < x \le 1$. Wegen $r'(x) < 0$ ist $r(x)$ monoton fallend, und es gilt damit $r(x) \le 1 = r(\Delta)$ im Bereich $\Delta \le x \le 1$. Hieraus folgt, dass sich die Graphen von $\ln(x)+1$ und $r(x)$ im Bereich $0 < \Delta < x < 1$ nur in einem Punkt schneiden, womit gezeigt ist, dass $g_2(a)$ in $-1+\Delta(a) < a < 0$ genau ein relatives Minimum besitzt. Mit diesen Ergebnissen kann eine Oberschranke des relativen Fehlers nach (6.39) berechnet werden, wenn die fehlerhaften Argumente $\widetilde{a}$ durch (6.37) festgelegt sind. Weitere Einzelheiten findet man im Quelltext der Funktion `rel_lnp1_abs()` auf Seite 177.

Nach (6.38) betrachten wir jetzt den Bereich der Maschinenwerte $\widetilde{a}$, für den die exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten Zahlenbereich liegen. Nach (6.7) von Seite 150 ist dann eine Oberschranke des relativen Fehlers gegeben durch:

$$\left|\frac{\texttt{lnp1}(\widetilde{a}) - \ln(1+a)}{\ln(1+a)}\right| \le h(a) := \varepsilon_{dat,f} \cdot [1+\varepsilon(f)] + \varepsilon(f), \tag{6.41}$$

wobei $\varepsilon_{dat,f} = \varepsilon_{dat,f}(a)$ nach (6.35) definiert ist. Da in $h(a)$ die Funktion $\varepsilon_{dat,f}(a)$ mit dem positiven Faktor $[1+\varepsilon(f)]$ multipliziert wird, können die Aussagen bez. des Maximums von $\varepsilon_{dat,f}(a)$ von (6.36) direkt auf die Funktion $h(a)$ übertragen werden:

In jedem Intervall $A = [\underline{a}, \overline{a}]$, mit $0 \notin A$ und $1+\underline{a}-\Delta(a) > 0$, nimmt $h(a)$ sein Maximum auf dem Rand von A an, d.h. es gilt:

$$\max_{a\in A} h(a) = \max_{a\in[\underline{a},\overline{a}]} h(a), \quad \text{falls } \; 0 \notin A \;\text{ und }\; 1+\underline{a}-\Delta(a) > 0\,; \tag{6.42}$$

Ist insbesondere $\underline{a} > 0$, so ist die Funktion $h(a)$ monoton fallend und muss daher nur einmal ausgewertet werden. Genauere Einzelheiten findet man im nachfolgenden Quelltext der Funktion `rel_lnp1_abs()`, die vom Modul `abs_relh` exportiert wird.

```
void rel_lnp1_abs(const interval& A, const real& dela,
                  interval& R, real& rel)
// rel is an upper bound of the relative error
// |(lnp1(wa)-ln(1+a))/ln(1+a)| for all exact function arguments a
// in A. dela is the absolute error bound of the erroneous function
// arguments wa; |wa-a|<=dela for all a in A
// R is an inclusion of all function values ln(1+a) with a in A.
{
    interval At,D,z=interval(-dela,dela);
    At = A+z;  D = At+z;  // At = A_tilde
    if (Inf(D)<=-1 || 0<=D)
    {
        cerr << "rel_lnp1_abs(): Relative error doesn't exist!"
             << endl;   exit(1);
```

```
}
// Now for any erroneous wa in At the corresponding exact
// value a lies in D and it holds:  a <> 0, and 1+a-dela > 0;
real r,r1,r2,In,Su,one_pe;
one_pe = addu(1,eps_lnp1);  // (1+eps(f))
if (Disjoint(At,MR_l)) // intersection is empty:
{  // ln(1+wa) only in the normalized range;
    In = Inf(At);  Su = Sup(At);
    if (sign(In) > 0)
    {
        r = subd(In,dela);
        r1 = Help_lnp1_1(r,dela);
        r = mulu(r1,one_pe);
        rel = addu(r,eps_lnp1);
    }
    else // Sup(At)<0:
    {
        r = addu(Su,dela);
        r1 = Help_lnp1_1(r,dela);
        r = mulu(r1,one_pe);
        rel = addu(r,eps_lnp1);
        r = subd(In,dela); // r: smallest possible argument a
        r1 = Help_lnp1_1(r,dela);
        r = mulu(r1,one_pe);
        r1 = addu(r,eps_lnp1);
        if (r1>rel) rel = r1;
    }
}
else // intersection not empty:
{
// Calculating an relative upper bound rel only for the exact
// function values ln(1+wa) lying in the denormalized range:
    D = At & MR_l;  // D: not empty intersection of At and MR_l;
    In = Inf(D);  Su = Sup(D);
    if (sign(In)>0)
    { // eps_dat,f + minreal/ln(1+a) is monotonic decreasing
        r = subd(In,dela);
        // r: smallest possible exact function argument;
        r1 = Help_lnp1_1(r,dela);
```

```
        z = lnp1( interval(r) );
        r2 = Inf(z);
        r2 = divu(minreal,r2);
        rel = addu(r1,r2); // rel: upper bound if Inf(D)>0;

        if (Sup(At)>MinReal)
        {
            r = subd(succ(MinReal),dela);
            r1 = Help_lnp1_1(r,dela);
            r = mulu(r1,one_pe);
            r1 = addu(r,eps_lnp1);
            if (r1>rel) rel = r1;
        }
    }
    else
    {
        r = addu(Su,dela); // r: greatest possible argument;
        r1 = Help_lnp1_1(r,dela);
        z = abs( lnp1(interval(r)) );
        r2 = Inf(z);
        r2 = divu(minreal,r2);
        rel = addu(r1,r2);  // rel: right upper bound;

        r = subd(In,dela);
        // r: smallest possible exact function argument;
        r1 = Help_lnp1_1(r,dela);
        z = abs( lnp1(interval(r)) );
        r2 = Inf(z);
        r2 = divu(minreal,r2);
        r2 = addu(r1,r2);  // r2: left upper bound;
        if (r2>rel) rel = r2;

        In = Inf(At);
        if (In <= -MinReal)
        {
            r1 = Help_lnp1_1(In,dela);
            r = mulu(r1,one_pe);
            r1 = addu(r,eps_lnp1);

            r = addu(-MinReal,dela);
```

```
                r2 = Help_lnp1_1(r,dela);
                r = mulu(r2,one_pe);
                r2 = addu(r,eps_lnp1);
                if (r2>r1) r1 = r2;
                if (r1>rel) rel = r1;
            }
        }
    } // else
    R = lnp1(A);
}
```

Die im Quelltext benutzte Funktion `Help_lnp1_1(..)` berechnet zu einem exakten Funktionsargument a eine Oberschranke der in (6.35) definierten Funktion $\varepsilon_{dat,f}(a)$.

In nachfolgender Tabelle sind zu gegebenem Maschinenintervall `A` und zur gegebenen absoluten Fehlerschranke $\Delta(a) \leq$ `dela` $\in S(2,53)$ die mit der obigen Funktion `rel_lnp1_abs()` berechneten garantierten Oberschranken `rel` des relativen Fehlers $|(\texttt{lnp1}(\widetilde{a}) - \ln(1+a))/\ln(1+a)| \leq$ `rel` angegeben:

`A` $\in IR$	$\Delta(a) \leq$ `dela`	`rel`
$[1,2]$	$1 \cdot 10^{-14}$	$7.464296 \cdot 10^{-15}$
$[1,2]$	$1 \cdot 10^{-16}$	$3.229548 \cdot 10^{-16}$
$[1,2]$	0.0	$2.508201 \cdot 10^{-16}$
$[2,30]$	$1 \cdot 10^{-16}$	$2.811614 \cdot 10^{-16}$
$[-0.75,-0.5]$	$1 \cdot 10^{-16}$	$5.393591 \cdot 10^{-16}$
$[-0.75,-0.25]$	$1 \cdot 10^{-16}$	$7.142946 \cdot 10^{-16}$
$[-0.75,-0.25]$	0.0	$2.508201 \cdot 10^{-16}$
$[3.952525 \cdot 10^{-323}, 1.000483 \cdot 10^{-320}]$	2^{-1074}	0.583334
$[2.964393 \cdot 10^{-323}, 1.000483 \cdot 10^{-320}]$	2^{-1074}	1.500000
$[2.964393 \cdot 10^{-323}, 1.000483 \cdot 10^{-320}]$	0.0	0.250000
$[2.964393 \cdot 10^{-323}, 1.000483 \cdot 10^{-320}]$	2^{-1073}	existiert nicht

Die letzten vier Beispiele zeigen, dass der relative Fehler um viele Größenordnungen anwächst, wenn die Funktionswerte $\ln(1+a)$ in den denormalisierten Bereich fallen, und zwar auch dann, wenn $\Delta(a) = 0$ gewählt wird. Beachten Sie bitte weiter, dass $2^{-1074} =$ `minreal` die kleinste positive Zahl in $S(2,53)$ ist.

Abschätzung des relativen Fehlers bei gegebenem $\varepsilon(a)$

Für alle exakten Funktionsargumente $a \in A = [\underline{a}, \overline{a}]$ suchen wir jetzt bei gegebenem $\varepsilon(a)$ eine Oberschranke des relativen Fehlers $|\texttt{lnp1}(\widetilde{a}) - \ln(1+a)/\ln(1+a)|$. Nach (6.7) benötigt man dazu die Schranke $\varepsilon_{dat,f}$ des fortgepflanzten Datenfehlers, die sich nach Seite 149 unter der Voraussetzung $1 + a \cdot [1 + \varepsilon(a)] > 0$ und $0 \le \varepsilon(a) < 1$ wie folgt berechnet:

$$\begin{aligned} \varepsilon_{dat,f}(a) &= |a| \cdot \varepsilon(a) \cdot \left| \frac{1}{(1 + a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]) \cdot \ln(1+a)} \right| \\ &= \begin{cases} h_1(a) := \dfrac{a \cdot \varepsilon(a)}{(1 + a \cdot [1 - \varepsilon(a)]) \cdot \ln(1+a)} & \text{falls } \; a > 0 \\ h_2(a) := \dfrac{a \cdot \varepsilon(a)}{(1 + a \cdot [1 + \varepsilon(a)]) \cdot \ln(1+a)} & \text{falls } \; a < 0 \end{cases} \end{aligned} \tag{6.43}$$

dabei ist $h_1(a)$ unter der Voraussetzung $1 - \varepsilon(a) \ge \frac{1}{2} \iff 0 \le \varepsilon(a) \le \frac{1}{2}$ streng[5] monoton fallend und ebenso $h_2(a)$, falls $1 + a \cdot [1 + \varepsilon(a)] > 0$.

Zum Beweis der Monotonie von $h_1(a)$ zeigen wir $h_1'(a) < 0$. Ganz elementare Rechnungen liefern mit $\delta := 1 - \varepsilon(a) > 0$:

$$h_1'(a) < 0 \quad \Longleftrightarrow \quad (1+a) \cdot \ln(1+a) < a + \delta \cdot a^2 \tag{6.44}$$

Den Beweis der rechten Ungleichung in (6.44) führen wir in zwei Schritten:

1. **Sei $0 < a < 1$** Die Reihenentwicklung von $(1+a) \cdot \ln(1+a)$ liefert:

$$\begin{aligned} (1+a) \cdot \ln(1+a) &= a + \frac{a^2}{2} - \frac{a^3}{6} + \frac{a^4}{12} - + \cdots \\ &< a + \frac{a^2}{2} \le a + \delta \cdot a^2 \end{aligned}$$

   und die letzte Ungleichung und damit auch (6.44) ist erfüllt für $\delta = 1 - \varepsilon(a) \ge \frac{1}{2}$.

2. **Sei $a \ge 1$** Zu zeigen ist wieder:

$$\begin{aligned} & (1+a) \cdot \ln(1+a) < a + \frac{a^2}{2} \\ \Longleftrightarrow \quad & \ln(1+a) < \frac{a \cdot (1 + a/2)}{1+a} \end{aligned} \tag{6.45}$$

   (6.45) gilt für $a = 1$, und weil (6.45) auch für die Ableitungen beider Terme richtig ist, gilt (6.45) für alle $a \ge 1$. Der Beweis bez. der Ableitungen ist trivial und bleibt daher dem Leser überlassen ∎

[5] $0 \le \varepsilon(a) \le \frac{1}{2}$ ist für die numerische Praxis mit $\varepsilon(a) \ll 1$ keine wirkliche Einschränkung!

Zum Nachweis der Monotonie von $h_2(a)$ zeigen wir $h_2'(a) < 0$, und einfache Rechnungen liefern mit $\alpha := 1 + \varepsilon(a) > 1$ unter der Voraussetzung[6] $1 + \alpha \cdot a > 0$:

$$h_2'(a) < 0 \quad \Longleftrightarrow \quad \ln(1+a) < a \cdot \frac{1+\alpha \cdot a}{1+a}, \quad \text{falls} \quad \frac{-1}{1+\varepsilon(a)} < a < 0. \tag{6.46}$$

Wegen $1 + \alpha \cdot a > 0$ und $\alpha := 1 + \varepsilon(a) > 1$ gilt auch $1 + a > 0$ und daher:

$$0 < \frac{1+\alpha \cdot a}{1+a} < 1 \quad \Longleftrightarrow \quad a \cdot \frac{1+\alpha \cdot a}{1+a} > a$$

und wegen der Äquivalenz in (6.46) gilt dann $h_2'(a) < 0$, falls $\ln(1+a) < a$. Wegen $-1 < a < 0$ ist diese letzte Ungleichung jedoch erfüllt und damit $h_2'(a)$ monoton fallend ∎

Nach (6.7) von Seite 150 ist jetzt wieder eine Fallunterscheidung nötig bez. der Lage der exakten Funktionswerte $\ln(1+\widetilde{a})$ im normalisierten bzw. denormalisierten Bereich. Es gilt:

$$-\texttt{MinReal} < \widetilde{a} \leq +\texttt{MinReal} \quad \Longrightarrow \quad |\ln(1+\widetilde{a})| \in [0, +\texttt{MinReal}) \tag{6.47}$$

$$-1 < \widetilde{a} \leq -\texttt{MinReal} \vee \widetilde{a} > +\texttt{MinReal} \quad \Longrightarrow \quad |\ln(1+\widetilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}] \tag{6.48}$$

Nach (6.47) betrachten wir zunächst den Bereich der Maschinenwerte $\widetilde{a}$, für den die exakten Funktionswerte $\ln(1+\widetilde{a})$ im denormalisierten Zahlenbereich liegen. Nach (6.7) von Seite 150 ist dann eine Oberschranke des relativen Fehlers gegeben durch:

$$\left| \frac{\texttt{lnp1}(\widetilde{a}) - \ln(1+a)}{\ln(1+a)} \right| \leq g(a) := \varepsilon_{dat,f} + \frac{\texttt{minreal}}{|\ln(1+a)|}, \tag{6.49}$$

wobei $\varepsilon_{dat,f} = \varepsilon_{dat,f}(a)$ nach (6.43) definiert ist. Man erkennt am letzten Summand rechts in (6.49), dass selbst im Falle $\varepsilon(a) = 0$, d.h. $\varepsilon_{dat,f} = 0$ die Oberschranke $g(a)$ des relativen Fehlers mit $a \to 0$ beliebig anwächst. In der numerischen Praxis macht es daher keinen Sinn, im Bereich (6.47) einen relativen[7] Fehler zu berechnen, vergleichen Sie in diesem Zusammenhang auch die entsprechenden Ergebnisse in der Tabelle von Seite 180.

Eine Oberschranke des relativen Fehlers wird also nur im Bereich (6.48) berechnet, in dem die Funktionswerte $\ln(1+\widetilde{a})$ alle im normalisierten Bereich liegen. Nach (6.7) von Seite 150 gilt dann:

$$\left| \frac{\texttt{lnp1}(\widetilde{a}) - \ln(1+a)}{\ln(1+a)} \right| \leq S(a) := \varepsilon_{dat,f} \cdot [1 + \varepsilon(f)] + \varepsilon(f), \tag{6.50}$$

wobei $\varepsilon_{dat,f} = \varepsilon_{dat,f}(a)$ wieder nach (6.43) definiert ist. Für $S(a)$ gilt daher:

[6] Wir setzen zusätzlich $\varepsilon(a) > 0$ voraus, da sonst $h_2(a) \equiv 0$ gilt.

[7] Die Berechnung einer absoluten Fehlerschranke mit Hilfe der Funktionen `abs_lnp1_abs()` oder `abs_lnp1_rel()` ist dagegen durchaus sinnvoll und notwendig!

$$S(a) = \begin{cases} S_1(a) := \dfrac{a \cdot \varepsilon(a) \cdot [1 + \varepsilon(f)]}{(1 + a \cdot [1 - \varepsilon(a)]) \cdot \ln(1 + a)} + \varepsilon(f) & \text{falls } a > 0 \\ S_2(a) := \dfrac{a \cdot \varepsilon(a) \cdot [1 + \varepsilon(f)]}{(1 + a \cdot [1 + \varepsilon(a)]) \cdot \ln(1 + a)} + \varepsilon(f) & \text{falls } a < 0 \end{cases}$$

dabei wurde vorausgesetzt: $0 \leq \varepsilon(a) < 1$ und $1 + a \cdot [1 + \varepsilon(a)] > 0$. Verlangt man zusätzlich $0 \leq \varepsilon(a) \leq \frac{1}{2}$, so sind mit den Monotonieaussagen von $h_1(a),\ h_2(a)$ die Funktionen $S_1(a),\ S_2(a)$ in ihrem jeweiligen Definitionsbereich beide monoton fallend. Um bei der Auswertung von $S_1(a),\ S_2(a)$ auf der Maschine für den relativen Fehler wirklich garantierte Oberschranken zu erhalten, müssen die auftretenden Grundoperationen durch gerichtete Rundungen ersetzt werden. So ist z.B. im Nenner von $S_1(a)$ der Faktor $(1 + a \cdot [1 - \varepsilon(a)])$ wie folgt auszuwerten:

$$(1 +< a *< [1 -< \varepsilon(a)])$$

Dabei bedeutet $1 -< \varepsilon(a)$ die Rundung der exakten Differenz $1 - \varepsilon(a)$ zur nächstkleineren Rasterzahl[8], und diese Rundung wird in **C-XSC** mit Hilfe der Funktion `subd()` realisiert. Zusätzliche Informationen findet man dazu auf Seite 184 im Quelltext der Funktion `rel_lnp1_rel(...)`, die vom Modul `abs_relh` exportiert wird.

In nachfolgender Tabelle sind zu gegebenem Maschinenintervall `A` und zu gegebener relativen Fehlerschranke $\varepsilon(a) \leq$ `epsa` $\in S(2, 53)$ die mit Hilfe der Funktion `rel_lnp1_rel()` berechneten garantierten Oberschranken `rel` des relativen Fehlers $|(\texttt{lnp1}(\widetilde{a}) - \ln(1 + a))/\ln(1 + a)| \leq$ `rel` angegeben:

`A` $\in IR$	$\varepsilon(a) \leq$ `epsa`	`rel`
$[1, 2]$	$1 \cdot 10^{-16}$	$3.229548 \cdot 10^{-16}$
$[1, 2]$	0.0	$2.508201 \cdot 10^{-16}$
$[2, 20]$	$1 \cdot 10^{-16}$	$3.115027 \cdot 10^{-16}$
$[10^{-60}, 10^{-50}]$	$1 \cdot 10^{-16}$	$3.508201 \cdot 10^{-16}$
$[-10^{-50}, -10^{-60}]$	$1 \cdot 10^{-16}$	$3.508201 \cdot 10^{-16}$
$[-10^{-50}, -10^{-60}]$	$1 \cdot 10^{-17}$	$2.608201 \cdot 10^{-16}$
$[-0.75, -0.5]$	$1 \cdot 10^{-16}$	$4.672243 \cdot 10^{-16}$
$[-0.75, -0.5]$	0.5	existiert nicht
$[-0.25, +0.25]$	0.0	existiert nicht

[8] Falls $1 - \varepsilon(a)$ selbst schon eine Rasterzahl ist, gilt: $1 -< \varepsilon(a) = 1 - \varepsilon(a)$.

Beachten Sie bitte, dass im vorletzten Beispiel die mit $\varepsilon(a) = \frac{1}{2}$ formulierte Bedingung $1 + a \cdot [1 + \varepsilon(a)] > 0$ von Seite 181 mit $a = -0.75$ nicht erfüllt ist und damit die Funktion `rel_lnp1_rel()` mit der entsprechenden Fehlermeldung abgebrochen wird. Ganz entsprechend wird die gleiche Funktion mit dem Intervall $A = [-0.25, +0.25]$ des letzten Beispiels abgebrochen, da die Funktionswerte $\ln(1+a)$ jetzt teilweise im denormalisierten Bereich liegen und damit viel zu große relative Fehlerschranken erzeugen würden.

```
void rel_lnp1_rel(const interval& A, const real& epsa,
                  interval& R, real& rel)
// rel is an upper bound of the relative error
// |(lnp1(wa)-ln(1+a))/ln(1+a)| for all exact function arguments a
// in A. epsa is the relative error bound of the erroneous function
// arguments wa; |wa-a|<=|a|*epsa for all a in A.
// R is an inclusion of all function values ln(1+a) with a in A.
{   real one_peps = addu(1,epsa),
         one_meps = subd(1,epsa),r,r1,abs_a,ar;
    interval At=A*interval(one_meps,one_peps); // At: A_tilde
    if (epsa>0.5 || Inf(At)<=-1 || Disjoint(At,MR_1)==0)
    { cerr << "rel_lnp1_rel(): Relative error doesn't exist!"
           << endl;  exit(1);}
    // All function values ln(1+wa) lie in the normalized range
    r = addu(1,eps_lnp1);
    r = mulu(r,epsa);  // r = epsa*(1+eps_lnp1)
    ar = Inf(A);
    if (sign(ar)>0) { r1 = one_meps;  abs_a = ar; }
    else { r1 = one_peps;  abs_a = -ar; }
    r = mulu(r,abs_a); // r: numerator
    r1 = muld(ar,r1);     r1 = addd(1,r1);
    At = abs( lnp1(interval(ar)) );
    ar = Inf(At);  ar = muld(r1,ar);  ar = divu(r,ar);
    rel = addu(ar,eps_lnp1); // relative error bound
    R = lnp1(A);
} // rel_lnp1_rel
```

Weitere Funktionen:

Im Modul `abs_relh` sind noch die folgenden Funktionen implementiert:

```
void rel_acosh_abs(const interval& A, const real& dela,
                                      interval& R, real& rel);
void rel_acosh_rel(const interval& A, const real& epsa,
                                      interval& R, real& rel);
```

Kapitel 7

Approximationsfehler

In diesem Kapitel werden Formeln hergeleitet, mit denen man bei Approximationen $f(x) \approx g(x)$ eine Schranke für den absoluten oder relativen Gesamtfehler angeben kann [6],[19]. Solche Approximationen sind für effektive Funktionsalgorithmen oft notwendig, wenn auf der Maschine die Auswertung von $f(x)$ im Vergleich zu $g(x)$ viel zu aufwendig ist. In der Praxis ist $g(x)$ entweder eine abgebrochene Taylorreihe oder eine rationale Funktion. Für ein vorgegebenes Argumentintervall A sind die Schranken $\Delta(app), \varepsilon(app)$ des absoluten bzw. relativen Fehlers wie folgt definiert:

$$|f(x) - g(x)| \leq \Delta(app), \qquad \left|\frac{f(x) - g(x)}{f(x)}\right| \leq \varepsilon(app) \quad \forall x \in A;$$

Beispiele zur Berechnung der Approximationsfehlerschranken $\Delta(app), \varepsilon(app)$ findet man u.a. in [5],[19] und in diesem Buch auf den Seiten 215,218,219,253,297,316,331.

7.1 Abschätzung des Gesamtfehlers

Bezeichnen wir mit $\widetilde{g}(\widetilde{x})$ die Maschinenauswertung von $g(x)$ für eine Maschinenzahl $\widetilde{x} \in A$, so ist eine Schranke $\varepsilon(g)$ des relativen Auswertefehlers definiert durch:

$$\left|\frac{g(\widetilde{x}) - \widetilde{g}(\widetilde{x})}{g(\widetilde{x})}\right| \leq \varepsilon(g) \quad \forall \widetilde{x} \in A,$$

und der relative Gesamtfehler lässt sich wie folgt abschätzen:

$$\begin{aligned}\left|\frac{f(\widetilde{x}) - \widetilde{g}(\widetilde{x})}{f(\widetilde{x})}\right| &\leq \left|\frac{f(\widetilde{x}) - g(\widetilde{x}) + g(\widetilde{x}) - \widetilde{g}(\widetilde{x})}{f(\widetilde{x})}\right| \\ &\leq \left|\frac{f(\widetilde{x}) - g(\widetilde{x})}{f(\widetilde{x})}\right| + \left|\frac{g(\widetilde{x}) - \widetilde{g}(\widetilde{x})}{g(\widetilde{x})}\right| \cdot \left|\frac{g(\widetilde{x})}{f(\widetilde{x})}\right|\end{aligned}$$

Mit den schon definierten relativen Fehlerschranken $\varepsilon(app), \varepsilon(g)$ folgt daher:

$$\begin{aligned}\left|\frac{f(\widetilde{x})-\widetilde{g}(\widetilde{x})}{f(\widetilde{x})}\right| &\leq \varepsilon(app)+\varepsilon(g)\cdot\left|\frac{g(\widetilde{x})-f(\widetilde{x})+f(\widetilde{x})}{f(\widetilde{x})}\right| \\ &\leq \varepsilon(app)+\varepsilon(g)\cdot\left(\left|\frac{g(\widetilde{x})-f(\widetilde{x})}{f(\widetilde{x})}\right|+1\right) \\ &\leq \varepsilon(app)+\varepsilon(g)\cdot[1+\varepsilon(app)] \quad \forall\,\widetilde{x}\in A.\end{aligned}$$

Bei Berücksichtigung eines Approximationsfehlers gilt damit für die Abschätzung des relativen Gesamtfehlers:

$$\boxed{\left|\frac{f(\widetilde{x})-\widetilde{g}(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f) := \varepsilon(app)+\varepsilon(g)\cdot[1+\varepsilon(app)] \quad \forall\,\widetilde{x}\in A} \tag{7.1}$$

und für die Abschätzung des absoluten Fehlers erhält man mit den Fehlerschranken $|f(\widetilde{x})-g(\widetilde{x})|\leq\Delta(app)$ und $|g(\widetilde{x})-\widetilde{g}(\widetilde{x})|\leq\Delta(g)$ ganz analog:

$$|f(\widetilde{x})-\widetilde{g}(\widetilde{x})| = |(f(\widetilde{x})-g(\widetilde{x}))+(g(\widetilde{x})-\widetilde{g}(\widetilde{x}))| \leq \Delta(app)+\Delta(g), \quad \text{d.h.}$$

$$\boxed{|f(\widetilde{x})-\widetilde{g}(\widetilde{x})| \leq \Delta(f) := \Delta(app)+\Delta(g) \quad \forall\,\widetilde{x}\in A} \tag{7.2}$$

Mit (7.1) und (7.2) sind wir damit in der Lage, bei einer Approximation $f(x)\approx g(x)$ neben dem Auswertefehler auch den Approximationsfehler bei der Abschätzung des Gesamtfehlers zu berücksichtigen.

7.1.1 $T(x) := h(x)\cdot f(x),\ f(x)\approx g(x)$

Ein vorgegebener reeller Term $T(x) := h(x)\cdot f(x)$ ist auf der Maschine auszuwerten, wobei $f(x)$ aus Laufzeitgründen für alle $x\in A$ durch g(x) approximiert wird. Die folgenden Einschließungsintervalle H, G werden als bekannt vorausgesetzt:

$$h(x)\in H \quad \text{und} \quad g(x)\in G \quad \forall x\in A;$$

Bekannt seien außerdem die Schranken $\Delta(h), \Delta(g)$ und $\Delta(app)$ der absoluten Auswertefehler und des absoluten Approximationsfehlers:

$$|h(\widetilde{x})-\widetilde{h}(\widetilde{x})|\leq\Delta(h), \quad |g(\widetilde{x})-\widetilde{g}(\widetilde{x})|\leq\Delta(g), \quad |f(x)-g(x)|\leq\Delta(app) \quad \forall\widetilde{x}, x\in A;$$

Um mit $\widetilde{T}(\widetilde{x}) := \widetilde{h}(\widetilde{x}) \odot \widetilde{f}(\widetilde{x})$ und $\widetilde{f}(\widetilde{x}) := \widetilde{g}(\widetilde{x})$ bei der automatischen Fehlerabschätzung z.B. eine Schranke $\Delta(T)$ des absoluten Fehlers

$$|T(\widetilde{x}) - \widetilde{T}(\widetilde{x})| \leq \Delta(T)$$

berechnen zu können, benötigen wir $\forall \widetilde{x} \in A$:

- eine Einschließung F der exakten Funktionswerte $f(x) \in F = ?$
- eine Oberschranke $\Delta(f)$ des absoluten Fehlers $|f(\widetilde{x}) - \widetilde{g}(\widetilde{x})| \leq \Delta(f) = ?$

Für die Schranke $\Delta(f)$ erhält man nach (7.2) von Seite 186 direkt das Ergebnis

$$\Delta(f) := \varepsilon(app) + \varepsilon(g) \quad \forall \widetilde{x} \in A;$$

Zur Berechnung von F beachten wir:

$$\begin{gathered} f(x) = g(x) + \delta_{app}, \quad \text{mit } |\delta_{app}| \leq \Delta(app) \quad \rightsquigarrow \\ \delta_{app} \in [-\Delta(app), +\Delta(app)] =: D(app) \quad \forall x \in A, \end{gathered}$$

und wegen $g(x) \in G$ ergibt sich daraus direkt die Einschließung für $f(x)$:

$$f(x) \in F := G + D(app) \quad \forall x \in A;$$

Die gesuchte Schranke $\Delta(T)$ lässt sich dann nach Tabelle 2.4 von Seite 40 durch den folgenden Funktionsaufruf

$$\texttt{abs\_mulh1}(H, \Delta(h), F, \Delta(f), R, \Delta(T))$$

berechnen, wobei R eine Einschließung aller exakten Termwerte $T(x)$ ist. Beachten Sie bitte, dass die Berechnung der Parameter F und $\Delta(f)$ bei der automatischen Fehlerabschätzung vor dem obigen Funktionsaufruf eigens zu programmieren ist. Ein praktisches Beispiel findet man dazu auf Seite 331.

7.2 Approximation mit Taylorpolynomen

Bei den allermeisten Standardfunktionen lässt sich nach einer geeigneten Argumentreduktion die eigentliche Funktionswertberechnung zurückführen auf die Auswertung der entsprechenden Taylorreihe in der unmittelbaren Umgebung des Ursprungs. Da diese Umgebungen Intervalle mit kleinen Durchmessern sind, können die exakten Funktionswerte durch Taylorpolynome mit nur geringer Ordnung N schon sehr gut approximiert werden:

$$f(x) \approx P_N(x) := \sum_{k=0}^{N} a_k \cdot x^k, \quad N = \ldots, 3, 4, 5, 6, \ldots;$$

Für eine korrekte Fehlerabschätzung muss neben dem Auswertefehler des Polynoms $P_N(x)$ natürlich auch der zugehörige Approximationsfehler berücksichtigt werden, der durch das Abschneiden der entsprechenden Taylorreihe entsteht. Die anzuwendenden Formeln sind in (7.1) und (7.2) auf Seite 186 angegeben.

In diesem Abschnitt stellen wir die Hilfsmittel vor, mit denen bei den Standardfunktionen die Approximationsfehler berechnet werden [6, 19]. Im Wesentlichen sind dies das Majorantenkriterium, die geometrische Reihe und das auf Leibniz zurückgehende Konvergenzkriterium für alternierende Reihen, das hier für die Anwendung formuliert werden soll:
Ein alternierende Reihe

$$s := \sum_{k=0}^{\infty} (-1)^k \cdot r_k, \quad r_k \geq 0,$$

deren absolute Reihenglieder r_k eine Nullfolge bilden, ist konvergent. Der Reihenwert s wird durch die Partialsumme $s_N := \sum_{k=0}^{N} (-1)^k \cdot r_k$ bis auf einen Fehler approximiert, der höchstens so groß ist wie der Betrag des ersten weggelassenen Summanden:

$$\left| s - \sum_{k=0}^{N} (-1)^k \cdot r_k \right| \leq r_{N+1} \tag{7.3}$$

Es gilt zusätzlich, dass s zwischen je zwei aufeinander folgenden Partialsummen s_k und s_{k+1} liegt, und mit $r_k \geq 0$ gilt dann:

$$s \in [s_{k+1}, s_k], \quad k = 0, 1, 2, \ldots \tag{7.4}$$

In den nachfolgenden Unterabschnitten wird je ein Beispiel zur Anwendung der geometrischen Reihe bzw. zur Anwendung des Leibniz-Kriteriums angegeben. Für die Funktionen $\sqrt{x+1}-1$, $\sqrt{x^2-1}$, $\sqrt{1-x^2}$ findet man Abschätzungen des relativen Approximationsfehlers auf den Seiten 216, 297, 316.

7.2.1 Abschätzung mit der geometrischen Reihe

Die Umkehrfunktion des hyperbolischen Tangens besitzt die folgende Taylorreihe:

$$\operatorname{artanh}(x) := \sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot x^{2k+1} = x \cdot P_\infty(x^2), \quad |x| < 1;$$

Approximiert man $\operatorname{artanh}(x)$ durch das Polynom $x \cdot P_N(x^2) := x \cdot \sum_{k=0}^{N} \frac{1}{2k+1} \cdot (x^2)^k$, so gilt für den relativen Approximationsfehler die Abschätzung:

$$\begin{aligned}
\left| \frac{\operatorname{artanh}(x) - x \cdot P_N(x^2)}{\operatorname{artanh}(x)} \right| &= \left| \frac{\displaystyle\sum_{k=N+1}^{\infty} \frac{1}{2k+1} \cdot x^{2k}}{\displaystyle\sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot x^{2k}} \right| \\
&\leq \frac{\frac{x^{2N+2}}{2N+3} \cdot (1 + x^2 + x^4 + \ldots)}{\displaystyle\sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot x^{2k}} = \frac{\frac{x^{2N+2}}{2N+3} \cdot \frac{1}{1-x^2}}{\displaystyle\sum_{k=0}^{\infty} \frac{1}{2k+1} \cdot x^{2k}} \\
&\leq \frac{x^{2N+2}}{2N+3} \cdot \frac{1}{1-x^2} =: \varepsilon(app; x, N), \quad |x| < 1;
\end{aligned}$$

Damit hat man mit $\varepsilon(app; x, N)$ einen in x monoton wachsenden geschlossenen Ausdruck, mit dem man den relativen Approximationsfehler einfach berechnen kann. Ist mit $0 < \tau < 1$ das Approximationsintervall gegeben durch $x \in [-\tau, +\tau]$, so erhält man eine garantierte Oberschranke des relativen Approximationsfehlers, wenn man zunächst τ durch das Intervall `ti = interval(`τ, τ`)` einschließt und danach $\varepsilon(app; x, N)$ intervallmäßig auswertet, indem man x formal durch das Intervall `ti` ersetzt und den so entstandenen Intervallausdruck auswertet. Ist dann `epsi` das Ergebnisintervall, so ist die Oberschranke des relativen Approximationsfehlers gegeben durch `Sup(epsi)`, d.h. $\varepsilon(app; x, N) \leq$ `Sup(epsi)` $\forall x \in [-\tau, +\tau]$. Ist in der Praxis ein Höchstwert des Approximationsfehlers vorgegeben, so muss der Polynomgrad N nur hinreichend groß gewählt zu werden, um diesen Höchstwert zu unterschreiten, da $\varepsilon(app; \tau, N)$ mit wachsendem N beliebig klein wird.

7.2.2 Abschätzung mit dem Leibniz-Kriterium

Die Umkehrfunktion des hyperbolischen Sinus besitzt die folgende Taylorreihe:

$$\operatorname{arsinh}(x) = \sum_{k=0}^{\infty} c_k \cdot x^{2k+1} =: x \cdot P_\infty(x^2), \quad |x| < 1;$$

$$c_0 = 1,\ c_1 = -\frac{1}{6},\ c_k := (-1)^k \cdot \frac{1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k-3) \cdot (2k-1)}{2 \cdot 4 \cdot 6 \cdot \ldots \cdot 2k \cdot (2k+1)},\ k \geq 2;$$

Für $|x| < 1$ ist damit $P_\infty(x^2) := \sum_{k=0}^{\infty} c_k \cdot (x^2)^k$ eine alternierende Leibnizreihe. Approximiert man $\operatorname{arsinh}(x)$ durch das Polynom $x \cdot P_N(x^2) := x \cdot \sum_{k=0}^{N} c_k \cdot (x^2)^k$, so gilt für den relativen Approximationsfehler die Abschätzung:

$$\begin{aligned} \left| \frac{\operatorname{arsinh}(x) - x \cdot P_N(x^2)}{\operatorname{arsinh}(x)} \right| &= \frac{\left| \sum_{k=N+1}^{\infty} c_k \cdot x^{2k} \right|}{|P_\infty(x^2)|} \leq \frac{|c_{N+1}| \cdot x^{2N+2}}{|P_\infty(x^2)|} \\ &\leq \frac{|c_{N+1}| \cdot x^{2N+2}}{1 - \frac{1}{6} \cdot x^2} =: \varepsilon(app; x, N), \quad |x| < 1; \end{aligned}$$

Beachten Sie bitte, dass die erste Abschätzung zustande kommt, weil obiger Zähler $\sum_{k=N+1}^{\infty} c_k \cdot x^{2k}$ genau der Fehler ist, der beim Abschneiden der Reihe $P_\infty(x^2)$ bei $k = N$ entsteht und weil dessen Betrag nach (7.3) durch $r_{N+1} := |c_{N+1}| \cdot x^{2N+2}$ abgeschätzt werden kann. Nach (7.4) gilt für $k = 0$ und $|x| < 1$:

$$0 < s_1 := 1 - \frac{1}{6}x^2 \leq P_\infty(x^2) \leq 1 =: s_0,$$

so dass wegen $|P_\infty(x^2)| = P_\infty(x^2)$ die obige zweite Abschätzung unmittelbar folgt. Zur praktischen Berechnung einer garantierten Oberschranke des relativen Approximationsfehlers mit Hilfe von $\varepsilon(app; x, N)$ gelten die gleichen Bemerkungen wie im Abschnitt 7.2.1.

7.3 Rationale Approximation

Im Gegensatz zu den Standardfunktionen, bei denen durch eine geeignete Argumentreduktion die eigentliche Funktionsauswertung stets in einer sehr engen Umgebung des Ursprungs stattfindet, müssen die Speziellen Funktionen der Mathematischen Physik in möglichst breiten Intervallen approximiert werden, da hier die genannte Argumentreduktion nicht möglich ist. Für eine möglichst kurze Laufzeit empfiehlt sich dabei die rationale Approximation, bei der die vorgegebene reelle Funktion $f(x)$ durch den Quotienten $P_M(x - x_1)/Q_N(x - x_1)$ zweier Polynome P_M, Q_N angenähert wird. Mit einem Computeralgebrasystem können dabei die Polynomkoeffizienten $\hat{p}_j, \hat{q}_j$ in hoher Genauigkeit so bestimmt werden, dass in einem vorgegebenen

Intervall $|x - x_1| \leq \eta$ die Betragsmaxima des relativen Approximationsfehlers zum Minimum gemacht werden. Bei dieser Approximation im Tschebyscheffschen Sinne besitzen dann im Idealfall die relativen Extrema des relativen Approximationsfehlers über dem ganzen Intervall $|x - x_1| \leq \eta$ die gleichen Beträge. Diese Beträge werden jedoch geringfügig differieren, wenn man in der Praxis gezwungen ist, die $\hat{p}_j, \hat{q}_j$ zur jeweils nächsten Rasterzahl $\dot{p}_j$ bzw. $\dot{q}_j$ des IEEE *double*-Formats zu runden. Die so definierten $\dot{p}_j, \dot{q}_j \in S(2, 53)$ fassen wir dann auf als die exakten Polynomkoeffizienten. Für eine Abschätzung des Gesamtfehlers nach (7.1) auf Seite 186 benötigt man daher für $f(x) \neq 0$ eine Abschätzung des relativen Approximationsfehlers:

$$\left| \frac{f(x) - \mathrm{P}_M(x - x_1)/\mathrm{Q}_M(x - x_1)}{f(x)} \right| \leq \varepsilon(app), \quad x \in z := \{x \in I\!R \mid |x - x_1| \leq \eta\}$$

Zur Berechnung einer garantierten Oberschranke $\varepsilon(app)$ wählt man eine Intervallzerlegung

$$z = \bigcup_j [z_j]$$

und wertet den obigen Quotienten intervallmäßig aus:

$$\varepsilon(app) := \max_j \left| \frac{f([z_j]) - \mathrm{P}_M([z_j] - x_1)/\mathrm{Q}_M([z_j] - x_1)}{f([z_j])} \right| \tag{7.5}$$

Auch bei einer sehr feinen Zerlegung des Approximationsintervalls z führt der Ansatz (7.5) jedoch nicht zum Ziel, denn je besser die Funktion $f(x)$ durch den Quotienten $\mathrm{P}_M(x - x_1)/\mathrm{Q}_M(x - x_1)$ approximiert wird, desto stärker macht sich die bekannte Auslöschung bei der Differenzbildung bemerkbar. Man beachte in diesem Zusammenhang, dass bei der Intervallsubtraktion der Durchmesser des Ergebnisintervalls definiert ist durch die Summe der Durchmesser der beteiligten Intervalloperanden, d.h. der Durchmesser des Ergebnisintervalls kann nicht kleiner werden als der maximale Durchmesser der beiden Operandenintervalle. Eine nach (7.5) berechnete Schranke ist zwar verlässlich, sie wird jedoch in der Regel den tatsächlichen Maximalwert des Approximationsfehlers um Größenordnungen übertreffen und ist damit in der Praxis unbrauchbar.

Nach W. Krämer [21] wird das Problem dadurch gelöst, dass man $f(x)$ mit Hilfe einer Staggered Taylor-Arithmetik in eine Taylorreihe entwickelt und die dann im Zähler auftretenden Polynome nicht einzeln auswertet, sondern vorher zu einem einzigen Polynom zusammenfasst, welches dann intervallmäßig ohne wesentliche Überschätzungen ausgewertet werden kann.

Zum besseren Verständnis des C-XSC Programms `Approx-Error`, mit dem zu einem vorgegebenen Intervall $|x - x_1| \leq \eta$ der absolute und relative Approximationsfehler berechnet werden kann, wird in den folgenden Abschnitten der in [21] beschriebene Lösungsweg nochmals aufgezeigt.

7.3.1 Numerische Auswertung der Schranken

Die Berechnung des absoluten und relativen Approximationsfehlers soll über dem Intervall $\mathtt{ab} := \{x \in I\!R \,|\, |x - x_1| \leq \eta\}$ erfolgen, wobei der Entwicklungspunkt x_1 in $S(2, 53)$ darstellbar sein muss. Die reelle Funktion

$$f : I\!R \supseteq D_f \longrightarrow I\!R$$

wird approximiert durch

$$f(x) \approx \frac{\mathrm{P}_M(x - x_1)}{\mathrm{Q}_N(x - x_1)} := \frac{\displaystyle\sum_{k=0}^{M} \dot{p}_k \cdot (x - x_1)^k}{\displaystyle\sum_{k=0}^{N} \dot{q}_k \cdot (x - x_1)^k}, \quad x \in \mathtt{ab} := \{x \in I\!R \,|\, |x - x_1| \leq \eta\},$$

Mit einem Computeralgebrasystem wurden die Polynomkoeffizienten $\hat{p}_k, \hat{q}_k$ zunächst in hoher Genauigkeit berechnet und anschließend zur jeweils nächsten IEEE-Zahl $\dot{p}_k$ bzw. $\dot{q}_k$ gerundet. Zur Vermeidung der bekannten Intervallüberschätzungen wird das Approximationsintervall $\mathtt{ab}$ in Z Teilintervalle $[z_j]$ zerlegt, und in jedem dieser $[z_j]$ wird dann der absolute und relative Approximationsfehler abgeschätzt.

$$\mathtt{ab} = \bigcup_j [z_j], \quad j = 1, 2, \ldots, Z;$$

Nach (7.5) ist dann $\varepsilon(app)$ das Maximum aller Oberschranken der relativen Approximationsfehler über allen Z Teilintervallen.

Wir betrachten jetzt eines der Z Teilintervalle $[z_j]$ mit dem Mittelpunkt $x_0 \in S(2, 53)$. Für die Funktion $f(x)$ gelte die Taylorentwicklung

$$f(x) = \sum_{k=0}^{\infty} s_k \cdot (x - x_0)^k, \quad |x - x_0| \leq \eta_0,$$

wobei die ersten Taylorkoeffizienten $s_k,\ k = 0, 1, \ldots, K$ mit Hilfe einer Taylor-Arithmetik eingeschlossen werden können. $f(x)$ wird in $|x - x_0| \leq \eta_0$ durch eine rationale Approximation der Form

$$f(x) \approx g(x) := \frac{\mathrm{p}_M(x - x_0)}{\mathrm{q}_N(x - x_0)} = \frac{\displaystyle\sum_{k=0}^{M} p_k \cdot (x - x_0)^k}{\displaystyle\sum_{k=0}^{N} q_k \cdot (x - x_0)^k}, \quad \mathrm{q}_N(x - x_0) \neq 0$$

angenähert, und wegen der verlangten Identitäten $\mathrm{P}_M(x-x_1) \equiv \mathrm{p}_M(x-x_0)$ und $\mathrm{Q}_N(x-x_1) \equiv \mathrm{q}_N(x-x_0)$ gelten die Beziehungen

$$p_k = \frac{1}{k!} \cdot \mathrm{P}_M^{(k)}(x-x_1)\Big|_{x=x_0} \quad \text{und} \quad q_k = \frac{1}{k!} \cdot \mathrm{Q}_N^{(k)}(x-x_1)\Big|_{x=x_0}, \tag{7.6}$$

wobei die auftretenden Taylorkoeffizienten mit Hilfe der C-XSC Taylor-Arithmetik eingeschlossen werden können. Weiter wird angenommen, dass $f(x)$ durch ein Taylorpolynom der Ordnung K approximiert wird und dass folgende Darstellung existiert:

$$f(x) = \sum_{k=0}^{K} s_k \cdot (x-x_0)^k + R_K(x) \tag{7.7}$$

Zur Abschätzung des auftretenden relativen Approximationsfehlers

$$\begin{aligned} \mathrm{err}(x) &:= \frac{g(x)-f(x)}{f(x)} = \frac{\dfrac{\mathrm{p}_M(x-x_0)}{\mathrm{q}_N(x-x_0)} - f(x)}{f(x)} \\ &= \frac{\mathrm{p}_M(x-x_0) - \mathrm{q}_N(x-x_0)\cdot f(x)}{\mathrm{q}_N(x-x_0)\cdot f(x)}, \quad |x-x_0| \le \eta_0, \end{aligned}$$

schreibt man den letzten Zähler mit Hilfe von (7.7) um in

$$\left\{ \mathrm{p}_M(x-x_0) - \mathrm{q}_N(x-x_0) \cdot \sum_{k=0}^{K} s_k \cdot (x-x_0)^k \right\} - \mathrm{q}_N(x-x_0) \cdot R_K(x)$$

Der eigentliche Trick besteht nun darin, den ersten Klammerausdruck als ein einziges Polynom

$$\mathrm{h}_L(x) := \sum_{k=0}^{L} r_k \cdot (x-x_0)^k$$

vom Grade $L := \max\{M, N+K\}$ aufzufassen und $\mathrm{h}_L(x)$ direkt auszuwerten. Die einzelnen Polynomkoeffizienten r_k können aus den bereits bekannten Polynomkoeffizienten p_k, q_k der Polynome $\mathrm{p}_M, \mathrm{q}_N$ sowie aus den s_k wie folgt berechnet werden:

$$r_k := \sum_{j=0}^{k} (s_j \cdot q_{k-j}) - p_k, \quad k = 0, 1, \ldots, L.$$

Auf einer Rechenanlage wird sich beim Berechnen der Koeffizienten r_k um so größere Auslöschung ergeben, je besser die rationale Approximationsfunktion $g(x)$ mit der zu approximierenden Funktion $f(x)$ übereinstimmt. Zur Lösung dieses Problems wird die in C-XSC implementierte Staggered Arithmetik verwendet. Genauer kann wie folgt vorgegangen werden:

- Berechne Einschließungen $[s_k]$ der Taylorkoeffizienten $s_k,\ k = 0, 1, \ldots, K$ der abgebrochenen Reihenentwicklung (7.7) mittels einer Staggered Arithmetik.
- Einschließungen $[r_k]$ der Koeffizienten des Polynoms $\mathrm{h}_L(x)$ können dann mittels
$$[r_k] := \sum_{j=0}^{k}([s_j] \cdot [q_{k-j}]) - [p_k], \quad k = 0, 1, \ldots, L$$
berechnet werden. Die Koeffizienteneinschließungen $[q_{k-j}]$, $[p_k]$ wurden dabei nach (7.6) schon vorher mit einer Staggered Taylorarithmetik bestimmt.

Unter Verwendung des Intervallpolynoms $[\mathrm{h}_L]$ kann nun eine sichere Oberschranke für den relativen Approximationsfehler der Approximation $g(x)$ über dem Intervall $|x - x_0| \leq \eta_0$ angegeben werden. Man findet zunächst

$$\begin{aligned}
\mathrm{err}(x) &:= \frac{\mathrm{p}_M(x-x_0) - \mathrm{q}_N(x-x_0) \cdot f(x)}{\mathrm{q}_N(x-x_0) \cdot f(x)} \\
&= \frac{\mathrm{p}_M(x-x_0) - \mathrm{q}_N(x-x_0) \cdot \sum_{k=0}^{K} s_k \cdot (x-x_0)^k}{\mathrm{q}_N(x-x_0) \cdot f(x)} - \frac{\mathrm{R}_K(x)}{f(x)} \\
&\in \frac{\sum_{k=0}^{L}[r_k] \cdot (x-x_0)^k}{\mathrm{q}_N(x-x_0) \cdot f(x)} - \frac{\mathrm{R}_K(x)}{f(x)},
\end{aligned}$$

und diese Inklusionsbeziehung gilt für alle x mit $|x - x_0| \leq \eta_0$. Kennt man eine obere Schranke τ für den Betrag $|\mathrm{R}_K(x)|$, gilt also

$$|\mathrm{R}_K(x)| \leq \tau \text{ für alle } x \text{ mit } |x - x_0| \leq \eta_0,$$

und bedeutet für ein gegebenes reelles Intervall A die Symbolik $|A|$ das Maximum aller Betragselemente von A, so ergibt sich die Ungleichung

$$\left| \frac{\sum_{k=0}^{L}[r_k] \cdot (x-x_0)^k}{\mathrm{q}_N(x-x_0) \cdot f(x)} - \frac{\mathrm{R}_K(x)}{f(x)} \right| \leq \frac{1}{|f(x)|} \cdot \left\{ \left| \frac{\sum_{k=0}^{L}[r_k] \cdot (x-x_0)^k}{\mathrm{q}_N(x-x_0)} \right| + \tau \right\} \tag{7.8}$$

Eine Einschließung von $f(x)$ ergibt sich aus (7.7) gemäß

$$f(x) \in \sum_{k=0}^{K}[s_k] \cdot (x-x_0)^k + [-\tau, +\tau], \quad |x - x_0| \leq \eta_0.$$

Dieser Ausdruck kann mittels „normaler“ Intervallarithmetik ohne große Auslöschung berechnet werden. Sollte der Wert 0 in diesem Intervall enthalten sein, so kann man auf diese Weise nur den absoluten Fehler der rationalen Approximation sicher nach oben abschätzen. Man beachte, dass man im Konvergenzbereich der Reihenentwicklung (7.7) die Fehlerschranke τ durch Erhöhung des Approximationsgrades K beliebig klein machen kann. Allerdings wächst bei zu großem K der Rechenaufwand zu stark, so dass in solchen Fällen der durch $2 \cdot \eta_0$ bestimmte Intervalldurchmesser verkleinert werden sollte.

Bezeichnet man die rechte Seite von (7.8) mit $\gamma(x)$, d.h.

$$\gamma(x) := \frac{1}{|f(x)|} \cdot \left\{ \left| \frac{\sum_{k=0}^{L} [r_k] \cdot (x - x_0)^k}{\mathrm{q}_N(x - x_0)} \right| + \tau \right\},$$

so ist man am Maximum

$$\mathrm{err}_j := \max\{\gamma(x) \,\big|\, |x - x_0| \leq \eta_0\}, \quad j = 1, 2, \ldots, Z$$

über dem Intervall $[z_j] := \{x \in I\!R \,\big|\, |x - x_0| \leq \eta_0$ interessiert. Um eine obere Schranke für $\gamma(x)$ zu bestimmen, kann man im einfachsten Fall überall im Ausdruck für $\gamma(x)$ die Punktgröße x durch das gesamte Intervall $[z_j]$ ersetzen. Symbolisch soll dieser Vorgang (intervallmäßige Auswertung von $\gamma(x)$) durch die Schreibweise

$$\gamma([z_j]) := \frac{1}{< f([z_j]) >} \cdot \left\{ \left| \frac{\sum_{k=0}^{L} [r_k] \cdot ([z_j] - x_0)^k}{\mathrm{q}_N([z_j] - x_0)} \right| + \tau \right\},$$

kenntlich gemacht. Dabei bezeichnet $< f([z_j]) >$ das Betragsminimum des Intervalls $f([z_j])$, und wir erhalten

$$\mathrm{err}_j \leq \gamma([z_j]), \quad j = 1, 2, \ldots, Z;$$

Sollte sich bei der intervallmäßigen Auswertung von $\gamma([z_j])$ eine zu große Überschätzung bemerkbar machen, so wird das Teilintervall $[z_j]$ mit festem Entwicklungspunkt x_0 und festen Polynomen $\mathrm{p}_M, \mathrm{q}_N$ in hinreichend viele, bis auf Randpunkte disjunkte, Teilintervalle $[\alpha_k]$ zerlegt:

$$[z_j] = \bigcup_k [\alpha_k], \quad k = 1, 2, \ldots, Z_2;$$

und man erhält direkt die Beziehungen:

$$\text{err}_j \le \gamma_j := \max_k\{\gamma[\alpha_k]\} \le \gamma([z_j]), \quad k = 1, 2, \ldots, Z_2; \tag{7.9}$$

$$\left| \frac{f(x) - \dfrac{\mathrm{P}_M(x - x_1)}{\mathrm{Q}_N(x - x_1)}}{f(x)} \right| \le \varepsilon(app) := \max_j\{\gamma_j\}, \quad j = 1, 2, \ldots, Z; \tag{7.10}$$

Für alle $x \in z := \{x \in I\!R \,\big|\, |x - x_1| \le \eta\}$ kann damit eine garantierte Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers ohne wesentliche Auslöschung berechnet werden.

7.3.2 Das Programm `Approx-Error`

Mit Hilfe des C-XSC Programms `Approx-Error` kann zu einer vorgegebenen reellen Zielfunktion

$$f : I\!R \supseteq D_f \longrightarrow I\!R$$

eine garantierte Oberschranke des absoluten und relativen Approximationsfehlers berechnet werden, wenn $f(x)$ im vorgegebenen Intervall `ab` $:= [a, b],\ a, b \in S(2, 53)$ mit dem Entwicklungspunkt $x_1 \in [a, b],\ x_1 \in S(2, 53)$ durch die folgende rationale Funktion approximiert wird:

$$f(x) \approx \frac{\mathrm{P}_M(x - x_1)}{\mathrm{Q}_N(x - x_1)} := \frac{\displaystyle\sum_{k=0}^{M} \dot{p}_k \cdot (x - x_1)^k}{\displaystyle\sum_{k=0}^{N} \dot{q}_k \cdot (x - x_1)^k}$$

Die als exakt anzusehenden Polynomkoeffizienten $\dot{p}_k, \dot{q}_k \in S(2, 53)$ werden dadurch gewonnen, dass man die zunächst mit einem Algebrasystem in hoher Genauigkeit berechneten Dezimalwerte $\hat{p}_k, \hat{q}_k$ anschließend zur jeweils nächsten Rasterzahl des IEEE-Systems rundet. Diese Rundung erfolgt automatisch nach dem Programmstart, wenn aus der Textdatei `APPR-Dat.txt` die folgenden dezimalen Daten gelesen werden:

```
M N x1 p0 p1 p2 ... pM q0 q1 q2 ... qN
```

Die obigen dezimalen Werte sind in `APPR-Dat.txt` nur durch Leerzeichen oder mit der RETURN-Taste zu trennen. Beim Programmstart müssen sich `Approx-Error` und `APPR-Dat.txt` im gleichen Verzeichnis befinden. Die Dezimalwerte $\hat{p}_k, \hat{q}_k$, die wir oben mit `p0 p1 ...` bzw. mit `q0 q1 ...` bezeichnet haben, sollten in `APPR-Dat.txt`

mit mindestens 19 oder 20 signifikanten dezimalen Stellen angegeben werden, damit bei der Rundung zur jeweils nächsten IEEE-Zahl der Rundungsfehler minimal bleibt. Beachten Sie bitte, dass diese Rundungsfehler nicht zu berücksichtigen sind, da wir die gerundeten IEEE-Zahlen $\dot{p}_k, \dot{q}_k$ als exakte Polynomkoeffizienten betrachten.

Die zu approximierende Zielfunktion $f(x)$ muss im Programm `Approx-Error.cpp` in der Funktion `l_itaylor STD(const l_itaylor& x){...}` definiert werden, dabei
können die Standardfunktionen aus folgender Tabelle beliebig verschachtelt und mit den vier Grundoperationen verknüpft werden.

Aufruf	Funktion	Aufruf	Funktion	Aufruf	Funktion
`sqr(x)`	x^2	`ln(x)`	$\ln(x)$	`asin(x)`	$\arcsin(x)$
`sqrt(x)`	$\sqrt{x}$	`lnp1(x)`	$\ln(1+x)$	`acos(x)`	$\arccos(x)$
`sqrt(x,n)`	$\sqrt[n]{x}$	`sin(x)`	$\sin(x)$	`atan(x)`	$\arctan(x)$
`sqrt1px2(x)`	$\sqrt{1+x^2}$	`cos(x)`	$\cos(x)$	`acot(x)`	$\operatorname{arccot}(x)$
`sqrtp1m1(x)`	$\sqrt{1+x}-1$	`tan(x)`	$\tan(x)$	`asinh(x)`	$\operatorname{arsinh}(x)$
`sqrt1mx2(x)`	$\sqrt{1-x^2}$	`cot(x)`	$\cot(x)$	`acosh(x)`	$\operatorname{arcosh}(x)$
`sqrtx2m1(x)`	$\sqrt{x^2-1}$	`sinh(x)`	$\sinh(x)$	`atanh(x)`	$\operatorname{artanh}(x)$
`pow(x,y)`	x^y	`cosh(x)`	$\cosh(x)$	`acoth(x)`	$\operatorname{arcoth}(x)$
`exp(x)`	e^x	`tanh(x)`	$\tanh(x)$		
`expm1(x)`	e^x-1	`coth(x)`	$\coth(x)$		

Tabelle 7.1: Standardfunktionen der Taylor-Arithmetik

Die Datentypen für die obigen Funktionsargumente `x,y,n` sind dabei folgendermaßen definiert: `l_itaylor x; int n; l_interval y;`

Nach dem Aufruf des Programms `Approx-Error` wir der Anwender aufgefordert, die folgenden Daten eizugeben:

1. Das Approximationsintervall $[a,b] \subseteq z := \{x \in I\!R \,\big|\, |x - x_1| \leq \eta\}$. Für das Intervall z wurden die dezimalen Polynomkoeffizienten $\hat{p}_k, \hat{q}_k$ mit Hilfe eines Algebrasystems berechnet.

2. Die Anzahl Z der Teilintervalle $[z_j]$, in die $[a,b]$ zur Vermeidung von Intervallüberschätzungen zu unterteilen ist.

3. Der Taylorpolynomgrad K zur Darstellung von $f(x)$ nach (7.7).

4. Der Parameter p, mit dem die Präzision der verwendeten Staggered Arithmetik festgelegt wird. $p = 3$ ist i.a. ein ausreichender Wert.

Am Anfang des Quelltextes `Approx-Error.cpp` können noch die Konstanten `Z1,Z2` festgelegt werden. Mit `Z1` kann eine weitere Intervallzerlegung vorgenommen werden,

um Intervallüberschätzungen bei der Abschätzung des Restgliedes $\mathrm{R}_k(x)$ in (7.7) zu vermeiden. Mit `Z2` wird jedes der `Z` Teilintervalle $[z_j]$ nochmals in `Z2` Teilintervalle zerlegt, um noch zu große Intervallüberschätzungen zu vermeiden. Um K nicht zu groß wählen zu müssen, d.h. also $K = \ldots 6, 7, 8, \ldots$, sollte man mit `Z2 = 50` die Teilintervallzahl `Z` möglichst klein wählen, um längere Laufzeiten zu vermeiden. Bei nicht zu breiten Approximationsintervallen $[a, b]$ erhält man mit $\mathtt{Z} = 700$ und $\mathtt{Z1} = 1$ i.a. schon recht gute, d.h. kleine Oberschranken des absoluten und relativen Approximationsfehlers. Wird für eine der Oberschranken der Wert $\mathtt{MaxReal} = 1.79 \ldots \cdot 10^{+308}$ ausgegeben, so konnte diese Schranke, z.B. wegen einer durch Intervallüberschätzung bedingten Division durch Null, nicht erfolgreich berechnet werden. Der Quelltext des Programms `Approx-Error.cpp` befindet sich im Anhang A ab Seite 395.

7.3.3 Beispiele

Mit den beiden folgenden Beispielen soll die korrekte Anwendung des Programms `Approx-Error` dokumentiert werden.

Im ersten Beispiel wird die für eine rationale Approximation eher atypische Zielfunktion $f(x) = e^x$ benutzt, die bekanntlich nach einer Argumentreduktion in einer engen Umgebung des Ursprungs durch ein Taylorpolynom der Ordnung 4 effektiv approximiert wird. Eine rationale Approximation der Exponentialfunktion ist daher aus Laufzeitgründen nicht zu empfehlen; als sehr einfache Programmanwendung hat $f(x) = e^x$ hier jedoch seine Berechtigung.

Im zweiten Beispiel wird als Zielfunktion die Gaußsche Fehlerfunktion $\mathrm{erf}(x)$ gewählt, die im Intervall $[0, 0.65]$ durch eine rationale Funktion zu approximieren ist. Dieses Beispiel zeigt für die speziellen Funktionen die grundsätzliche Vorgehensweise bei der Abschätzung des relativen Approximationsfehlers bei rationaler Approximation.

7.3.3.1 $f(x) = e^x$

Die Exponentialfunktion $f(x) = e^x$ soll im Intervall $[a, b] = [-1, +1]$ durch eine rationale Funktion $g(x) := \mathrm{P}_5(x)/\mathrm{Q}_5(x)$ mit dem Entwicklungspunkt $x_1 = 0$ approximiert werden:

$$e^x \approx g(x) := \frac{\mathrm{P}_5(x)}{\mathrm{Q}_5(x)} = \frac{\sum_{k=0}^{5} \dot{p}_k \cdot x^k}{\sum_{k=0}^{5} \dot{q}_k \cdot x^k}, \quad \dot{p}_k, \dot{q}_k \in S(2, 53); \tag{7.11}$$

Zu berechnen ist eine Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers

$$\left|\frac{f(x)-g(x)}{f(x)}\right| \leq \varepsilon(app) = ? \quad \forall x \in [-1,1]; \tag{7.12}$$

Mit dem Algebrasystem *Mathematica* berechnet man zunächst mit den Anweisungen

```
<< NumericalMath‘Approximations‘
   g[t_] := Exp[t];
   gmma2 =  GeneralMiniMaxApproximation[{t,g[t]},
     {t, {-1,+1}, 5, 5}, u, WorkingPrecision -> 50,
     Bias -> 0.25]
     app1[u_] := gmma2[[2,1]]
```

die jeweils 6 dezimalen Polynomkoeffizienten $\hat{p}_k, \hat{q}_k$

```
{(0.99999999999999999999999294274448294863515978539834
    0.49999999998745859522668444535673226082064341965 84u +
    0.11108342323038889822762175057174225918378564748223u^2 +
    0.01387504495931447511727575492556585490991443674 6806u^3 +
    0.00098929812146598095332666923870186490934248985007322u^4 +
    0.00003283969887589120985083743261105099632043401 6800885u^5)/
  (1 - 0.50000000000018851621199733726132103221582057468708u +
    0.11108342323111022657017091667814006056261554900056u^2 -
    0.01387504495946894066909407307029235403378961586 6426u^3 +
    0.00098929812148310305480541456597019149159328198910901u^4 -
    0.00003283969887674132794960514849086745188820325 2586099u^5),
  9.708252378905680194295682310187005781908652116028 7*10^-14}
```

in hoher Genauigkeit und den Schätzwert $9.7082523789\ldots \cdot 10^{-14}$ für den relativen Approximationsfehler. Mit einem Editor werden dann in die Textdatei `APPR-Dat.txt` folgende Werte eingetragen: $M = 5, N = 5, x_1 = 0$ und in ausreichender Genauigkeit die jeweils 6 Polynomkoeffizienten $\hat{p}_k, \hat{q}_k$. `APPR-Dat.txt` enthält damit die Einträge:

```
5 5
0.0
0.99999999999999999999999929427444829
0.49999999998745859522668444535673 2
0.11108342323038889822762175057174 22
1.38750449593144751172757549255658 54e-2
9.89298121465980953326669238701864 90e-4
3.28396988758912098508374326110509 96e-5
```

```
1.0
-0.50000000000018851621199733726 1321
+1.1108342323111022657017091667814006e-1
-1.3875044959468940669094073070292 35e-2
+9.8929812148310305480541456597019149e-4
-3.2839698876741327949605148490867 45e-5
```

Im Quelltext `Approx-Error.cpp` wird die Zielfunktion $f(x) = e^x$ in der Funktion `l_itaylor STD(const l_itaylor& x)` wie folgt definiert:

```
l_itaylor STD(const l_itaylor& x)
{   // Definition of the function to be approximated
    // by a rational function
    l_itaylor t;
    t = exp(x); // Definition of the exponential function
    return t;
}
```

Nach erfolgreicher Übersetzung wird der Anwender nach dem Programmstart aufgefordert, die folgenden Werte einzugeben:

```
Approximation intervall [a,b] = ?              [-1,+1]  RET
Z: Number of subintervals in [a,b];  Z = ?     1200     RET
f(x): Taylor expansion of degree K = ?         8        RET
Multiprecision parameter P = ?                 3        RET
```

Nach einigen Sekunden erhält man die folgende Bildschirmausgabe:

```
*   Error bound for the absolute error:  2.6385791630E-013
*   Error bound for the relative error:  9.7128170652E-014
```

Nach (7.12) ist damit eine garantierte Schranke des relativen Approximationsfehlers gegeben durch:

$$\left|\frac{f(x)-g(x)}{f(x)}\right| \leq \varepsilon(app) = 9.7128170652 \cdot 10^{-14} \quad \forall x \in [-1,1];$$

7.3.3.2 $f(x) = \mathrm{erf}(x)$

Die Gaußsche Fehlerfunktion ist definiert durch

$$\mathrm{erf}(x) := \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt, \quad x \in I\!R;$$

Wegen der Punktsymmetrie zum Ursprung kann man sich beschränken auf $x \geq 0$. Im Intervall $\boldsymbol{A} = [0, 0.65]$ soll $\mathrm{erf}(x)$ möglichst effektiv approximiert werden, d.h. die Laufzeit der verwendeten Approximationsfunktion soll möglichst optimal sein. Dazu wird $\boldsymbol{A}$ in zwei Teilintervalle $\boldsymbol{A_1} = [0, 10^{-10}]$ und $\boldsymbol{A_2} = [10^{-10}, 0.65]$ unterteilt. Im Bereich $\boldsymbol{A_1}$ benutzen wir die Reihenentwicklung

$$\mathrm{erf}(x) = \frac{2x}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n \cdot x^{2n}}{n!(2n+1)} = \frac{2x}{\sqrt{\pi}} \left[1 - \frac{x^2}{3} + \frac{x^4}{10} - + \cdots \right], \quad |x| < \infty;$$

Da diese Reihe für $|x| < 1$ eine Leibniz-Reihe ist, gilt bezüglich der Approximation

$$\mathrm{erf}(x) \approx G_1(x) := \frac{2}{\sqrt{\pi}} \cdot x$$

für den absoluten Fehler $r(x) := \mathrm{erf}(x) - 2x/\sqrt{\pi}$ nach (7.3) von Seite 188 die Abschätzung

$$|r(x)| \leq \frac{2x^3}{3 \cdot \sqrt{\pi}}$$

Zusammen mit

$$\mathrm{erf}(x) > \frac{2x}{\sqrt{\pi}} \cdot \left(1 - \frac{x^2}{3} \right)$$

ergibt sich dann für den relativen Approximationsfehler $\varepsilon_{\mathrm{app}}(x) := r(x)/\mathrm{erf}(x)$ die Abschätzung[1]:

$$|\varepsilon_{\mathrm{app}}(x)| \leq \frac{x^2}{3 - x^2} \leq \frac{10^{-20}}{3 - 10^{-20}} < 3.3334 \cdot 10^{-21} =: \varepsilon(\mathrm{app}), \quad \forall x \in \boldsymbol{A_1} = [0, 10^{-10}].$$

Wir kommen jetzt zur Abschätzung des relativen Approximationsfehlers im Intervall $\boldsymbol{A_2} = [10^{-10}, 0.65]$, in dem die Annäherung durch eine rationale Funktion $g(x)$ zur Anwendung kommt.

Bei der Approximation der speziellen Funktionen durch eine rationale Funktion $g(x) := \mathrm{P}_M(x - x_1)/\mathrm{Q}_N(x - x_1)$ sind i.a. **zwei** Approximationsschritte notwendig. Zunächst muss eine Hilfsfunktion $H(x)$ zur Approximation von $f(x)$ gefunden werden, welche durchaus recht kompliziert aufgebaut sein darf. Es wird dabei nur verlangt, dass $H(x)$ mit Hilfe von bereits implementierten Intervallfunktionen und Intervalloperatoren programmierbar ist, vgl. Tabelle 7.1 auf Seite 198. Die entsprechende Approximationsfehlerschranke $\varepsilon(\mathrm{app}, 1)$ muss dabei in der Regel analytisch, d.h. per Hand hergeleitet werden. Die Gewinnung der eigentlichen Approximationsfunktion $g(x)$ für die Implementierung der Ausgangsfunktion $f(x)$ erfolgt dann mit

[1] Der Term $x^2/(3 - x^2)$ ist dabei intervallmäßig auszuwerten, d.h. x ist durch ein Intervall `z` zu ersetzen, das 10^{-10} optimal einschließt: `interval z = interval(1E-10,1E-10);`

Hilfe eines Algebrasystems, wobei man die i.a. komplizierte Funktion $H(x)$ dort als Zielfunktion nur dann benutzen wird, wenn $f(x)$ im benutzten Algebrasystem nicht implementiert ist. Eine Schranke $\varepsilon(\text{app}, 2)$ für den Approximationsfehler der zweiten Approximation $H(x) \approx g(x)$ kann dann mit Hilfe des Programms `Approx-Error` automatisch bestimmt werden. Für die zwei Approximationen

$$f(x) \approx H(x) \approx g(x), \quad x \in [a, b] \tag{7.13}$$

wird jetzt mit Hilfe der als bekannt vorausgesetzten Schranken $\varepsilon(\text{app}, 1)$, $\varepsilon(\text{app}, 2)$ der relative Approximationsfehler bezüglich $f(x) \approx g(x)$ hergeleitet:

$$\begin{aligned}
\left|\frac{f(x) - g(x)}{f(x)}\right| &= \left|\frac{(f(x) - H(x)) + (H(x) - g(x))}{f(x)}\right|, \quad f(x) \neq 0 \\
&\leq \left|\frac{f(x) - g(x)}{f(x)}\right| + \left|\frac{H(x) - g(x)}{f(x)}\right| \\
&\leq \varepsilon(\text{app}, 1) + \left|\frac{H(x)}{f(x)}\right| \cdot \left|\frac{H(x) - g(x)}{H(x)}\right|, \quad H(x) \neq 0 \\
&\leq \varepsilon(\text{app}, 1) + \left|1 + \frac{-f(x) + H(x)}{f(x)}\right| \cdot \varepsilon(\text{app}, 2) \\
&\leq \varepsilon(\text{app}, 1) + (1 + \varepsilon(\text{app}, 1)) \cdot \varepsilon(\text{app}, 2);
\end{aligned}$$

Im Approximationsintervall $[a, b]$ gilt damit für die beiden Approximationen nach (7.13) die Abschätzung:

$$\boxed{\left|\frac{f(x) - g(x)}{f(x)}\right| \leq \varepsilon(\text{app}) := \varepsilon(\text{app}, 1) + (1 + \varepsilon(\text{app}, 1)) \cdot \varepsilon(\text{app}, 2);} \tag{7.14}$$

Bezüglich der Gaußschen Fehlerfunktion $f(x) = \text{erf}(x)$ bestimmen wir jetzt die **erste Approximationsfunktion:** $\text{erf}(x) \approx H(x), \quad x \in \boldsymbol{A_2} = [10^{-10}, 0.65]$.

Nach [1, Formel 7.1.6] gilt für $|x| < +\infty$ die Reihenentwicklung:

$$\text{erf}(x) = \frac{2x}{\sqrt{\pi}} \cdot e^{-x^2} \cdot \sum_{n=0}^{\infty} a_n \cdot x^{2n}, \qquad a_n := \frac{2^n}{1 \cdot 3 \cdot \ldots \cdot (2n+1)}; \tag{7.15}$$

Wegen $a_n / a_{n+1} < 1$ erhält man mit

$$\sum_{n=0}^{\infty} a_n \cdot x^{2n} = \sum_{n=0}^{N} a_n \cdot x^{2n} + \sum_{n=N+1}^{\infty} a_n \cdot x^{2n}$$

die Abschätzung

$$\begin{aligned}\sum_{n=N+1}^{\infty} a_n \cdot x^{2n} &= a_{N+1} \cdot x^{2(N+1)} \left[1 + \frac{a_{N+2}}{a_{N+1}} \cdot x^2 + \frac{a_{N+3}}{a_{N+1}} \cdot x^4 + \cdots \right] \\ &< a_{N+1} \cdot x^{2(N+1)} \sum_{n=0}^{\infty} x^{2n} = \frac{a_{N+1} \cdot x^{2(N+1)}}{1-x^2}, \quad |x| < 1;\end{aligned}$$

Mit der ersten Approximation

$$\operatorname{erf}(x) \approx H(x) := \frac{2x}{\sqrt{\pi}} \cdot e^{-x^2} \cdot \sum_{n=0}^{N} a_n \cdot x^{2n}$$

und wegen der durch die Reihenentwicklung gegebenen Ungleichung

$$\operatorname{erf}(x) \geq \frac{2x}{\sqrt{\pi}} \cdot e^{-x^2}, \quad x \geq 0$$

folgt dann für den relativen Approximationsfehler:

$$\begin{aligned}\varepsilon_{\text{app},1}(x) &:= \frac{\operatorname{erf}(x) - H(x)}{\operatorname{erf}(x)}; \\ |\varepsilon_{\text{app},1}(x)| &< \frac{2^{N+1} \cdot x^{2(N+1)}}{1 \cdot 3 \cdot \ldots \cdot (2N+3)} \cdot \frac{1}{1-x^2} \leq \varepsilon(\text{app},1), \quad |x| < 1; \end{aligned} \tag{7.16}$$

Für $N = 14$ gilt damit im Bereich $\boldsymbol{A_2} = [10^{-10}, 0.65]$

$$\boxed{\operatorname{erf}(x) \approx H(x) := \frac{2x}{\sqrt{\pi}} \cdot e^{-x^2} \cdot \sum_{n=0}^{14} a_n \cdot x^{2n}, \quad \varepsilon(\text{app},1) := 7.2149 \cdot 10^{-19}} \tag{7.17}$$

$\varepsilon(\text{app},1) = 7.2149 \cdot 10^{-19}$ ergibt sich dabei durch Intervallauswertung des monotonen Bruchs in (7.16), wobei x durch ein Intervall `z` zu ersetzen ist, das $x = 0.65$ optimal einschließt, d.h. `interval z = interval(0.65,0.65);`

Die Hilfsfunktion $H(x)$ könnte durchaus als Maschinenapproximation für $\operatorname{erf}(x)$ benutzt werden, jedoch wäre die Laufzeit wegen der auftretenden Exponentialfunktion und wegen des hohen Polynomgrades $N = 14$ sehr groß! Viel effektiver ist daher die Verwendung einer **zweiten Approximation** mit

$$\operatorname{erf}(x) \approx g(x) := x \cdot \frac{\mathrm{P}_4(x^2)}{\mathrm{Q}_4(x^2)}, \qquad \mathrm{P}_4(x^2) := \sum_{k=0}^{4} \dot{p}_k \cdot x^{2k}, \quad \mathrm{Q}_4(x^2) := \sum_{k=0}^{4} \dot{q}_k \cdot x^{2k};$$

Genauer müssen wir jedoch nach (7.13) die Approximation

$$H(x) \approx g(x) := x \cdot \frac{\mathrm{P}_4(x^2)}{\mathrm{Q}_4(x^2)}$$

betrachten und eine garantierte Oberschranke $\varepsilon(\mathrm{app}, 2)$ des zugehörigen relativen Approximationsfehlers bestimmen:

$$\left|\frac{H(x) - g(x)}{H(x)}\right| \leq \varepsilon(\mathrm{app}, 2) = ? \quad \forall x \in \boldsymbol{A_2} = [10^{-10}, 0.65]$$

Mit

$$h(x) := \frac{2}{\sqrt{\pi}} \cdot e^{-x^2} \cdot \sum_{n=0}^{14} a_n \cdot x^{2n} \qquad \text{gilt die Identität:} \tag{7.18}$$

$$\left|\frac{H(x) - g(x)}{H(x)}\right| \equiv \left|\frac{h(x) - \mathrm{P}_4(x^2)/\mathrm{Q}_4(x^2)}{h(x)}\right| \leq \varepsilon(\mathrm{app}, 2) = ? \tag{7.19}$$

so dass jetzt zur Berechnung von $\varepsilon(\mathrm{app}, 2)$ die Funktion $h(x) := H(x)/x$ benutzt wird. Die rationale Approximationsfunktion $\mathrm{P}_4(x^2)/\mathrm{Q}_4(x^2)$ muss jedoch bei Anwendung des Programms `Approx-Error` wie folgt umgeschrieben werden

$$\frac{\mathrm{P}_4(x^2)}{\mathrm{Q}_4(x^2)} \equiv \frac{\mathrm{P}_8(x)}{\mathrm{Q}_8(x)} = \frac{\sum_{k=0}^{8} \dot{p}_k \cdot x^k}{\sum_{k=0}^{8} \dot{q}_k \cdot x^k}, \qquad \dot{p}_k = \dot{q}_k = 0, \quad k = 1, 3, \ldots 7,$$

da in `Approx-Error` Polynome in x^p oder $(x - x_1)^p$ nur für $p = 1$ zulässig sind. Mit den Anweisungen

```
<< NumericalMath‘Approximations‘
   g[t_] := Erf[t]/t;
   gmma2 =  GeneralMiniMaxApproximation[{t^2,g[t]},
     {t, {-10^(-12),13/20}, 4, 4}, u, WorkingPrecision -> 50,
     Bias -> 0.25]
     app1[u_] := gmma2[[2,1]]
```

liefert das Algebrasystem *Mathematica* für $\mathrm{P}_4(x^2), \mathrm{Q}_4(x^2)$ die folgenden dezimalen Polynomkoeffizienten in hoher Genauigkeit:

```
{(1.12837916709551257009607645302673929775716672411 46 +
  0.13589488762727791499031957644772046895727149819996u +
```

```
   0.040325948853179525110061266675604716757990137045438lu^2 +
   0.001203393808630794604151433244863711264921852209117723781u^3 +
   0.000064925455648190438013449697736388391362506475547721244 3u^4)/
  (1 + 0.453767041780002559250927289795570377874372439713811u +
   0.086993622261538592835663911368056964287127003974977u^2 +
   0.008497173711686933258877662820529620834270092152453 6u^3 +
   0.000364915280629351078725564412701648783584257293417551u^4),
     3.367735386214503549932090881921543816140066566057972*10^-18}
```

zusammen mit dem Schätzwert $3.3677\ldots\cdot 10^{-18}$ für den relativen Approximationsfehler. Mit einem Editor werden dann in die Textdatei `APPR-Dat.txt` folgende Werte eingetragen: $M = 8, N = 8, x_1 = 0$ und in ausreichender Genauigkeit die jeweils 8 Polynomkoeffizienten $\hat{p}_k, \hat{q}_k$. `APPR-Dat.txt` enthält damit die nachfolgenden Daten:

```
8 8
0.0
1.128379167095512570096
0.0
1.358948876272779149903E-1
0.0
4.032594885317952511006E-2
0.0
1.203393808630794604151E-3
0.0
6.492545564819043801345E-5
1.0
0.0
4.537670417800025592509E-1
0.0
8.699362226153859283566E-2
0.0
8.497173711686933258877E-3
0.0
3.649152806293510787255E-4
```

Zur Berechnung von $\varepsilon(\text{app}, 2)$ muss im Quelltext `Approx-Error.cpp` nach (7.18) die Zielfunktion $h(x)$ in der Funktion `l_itaylor STD(const l_itaylor& x)` wie folgt definiert werden:

```
l_itaylor STD(const l_itaylor& x)
{   // Definition of the function to be approximated
```

```
    // by a rational function
    l_itaylor t,t1,x2;
    l_interval c = 4*li_pi4();  // c = pi;
    l_interval a14 = l_interval(16384);  // a14 = 2^(14)
    l_interval ak;

    for (int k=1;k<=14;k++) a14 = a14 / (2*k+1);
    // a14 = 2^(14) / (1*3*5* ... *29);

    x2 = sqr(x);  // x2 is a VARIABLE of type l_itaylor
    t1 = 2*exp(-x2)/sqrt(c);
    t  = const_l_itaylor(get_order(x),a14);
    // t is now a constant of type l_itaylor!
    ak = a14; // Start value of the recursion formula
    for (int k=13;k>=0;k--)  // Horner's scheme
    {
        ak = ak*(2*k+3)/2;   // Recursion formula for ak
        t = t*x2 + ak;
    }
    return t*t1;
}  // STD
```

Nach erfolgreicher Übersetzung von `Approx-Error` wird der Anwender nach dem Programmstart aufgefordert, die folgenden Werte einzugeben:

```
Approximation intervall [a,b] = ?              [1e-10,0.65]  RET
Z: Number of subintervals in [a,b];  Z = ?     800           RET
f(x): Taylor expansion of degree K = ?         8             RET
Multiprecision parameter P = ?                 2             RET
```

Nach einigen Sekunden erhält man die folgende Bildschirmausgabe:

```
*   Error bound for the absolute error:  1.5335469654E-017
*   Error bound for the relative error:  1.3590706152E-017
```

Nach (7.19) ist damit eine garantierte Schranke des relativen Approximationsfehlers gegeben durch:

$$\left|\frac{H(x)-g(x)}{H(x)}\right| \leq \varepsilon(\text{app},2) = 1.3590706152 \cdot 10^{-17} \quad \forall x \in \boldsymbol{A_2} = [10^{-10}, 0.65];$$

Mit Hilfe der bekannten Fehlerschranken $\varepsilon(\text{app},1),\ \varepsilon(\text{app},2)$ kann jetzt mit (7.13) und (7.14) der relative Approximationsfehler $\varepsilon(\text{app})$ berechnet werden. Durch Intervallauswertung von (7.14) erhält man

$$\text{erf}(x) \approx x \cdot \frac{\text{P}_4(x^2)}{\text{Q}_4(x^2)};\ \varepsilon(\text{app}) := 1.4313 \cdot 10^{-17}\ \forall x \in \boldsymbol{A_2} = [10^{-10}, 0.65] \tag{7.20}$$

Die Koeffizienten $\dot{p}_k, \dot{q}_k \in S(2,53)$ der Polynome $\text{P}_4(x^2), \text{Q}_4(x^2)$ sind die zur jeweils nächsten Rasterzahl gerundeten Dezimalwerte aus der Tabelle 7.2 von Seite 209.

k	$\dot{p}_k :=$ nearest(.)	$\dot{q}_k :=$ nearest(.)
0	$1.128379167095512570096 \cdot 10^{+0}$	$1.000000000000000000000 \cdot 10^{+0}$
1	$1.358948876272779149903 \cdot 10^{-1}$	$4.537670417800025592509 \cdot 10^{-1}$
2	$4.032594885317952511006 \cdot 10^{-2}$	$8.699362226153859283566 \cdot 10^{-2}$
3	$1.203393808630794604151 \cdot 10^{-3}$	$8.497173711686933258877 \cdot 10^{-3}$
4	$6.492545564819043801345 \cdot 10^{-5}$	$3.649152806293510787255 \cdot 10^{-4}$

Tabelle 7.2: Koeffizienten $\dot{p}_k, \dot{q}_k \in S(2,53)$ der Polynome $\mathrm{P}_4(x^2), \mathrm{Q}_4(x^2)$

Anmerkungen:

1. Beachten Sie bitte, dass das Programm `Approx-Error` nur dann richtig funktioniert, wenn die Polynome $\mathrm{P}_4(x^2), \mathrm{Q}_4(x^2)$ geschrieben werden als $\mathrm{P}_8(x), \mathrm{Q}_8(x)$, da die rational Approximationsfunktion in der Form $\mathrm{P}_M(x - x_1)/\mathrm{Q}_N(x - x_1)$ vorliegen muss. Vergleichen Sie dazu die Datei `APPR-Dat.txt` auf Seite 206.

2. Auf dem Rechner wird $\mathrm{erf}(x)$ in $\boldsymbol{A_2} = [10^{-10}, 0.65]$ durch $x \cdot \mathrm{P}_4(x^2)/\mathrm{Q}_4(x^2)$ approximiert. Bei der Berechnung der Oberschranke $\varepsilon(\mathrm{app}, 2)$ gilt für den zugehörigen relativen Approximationsfehler nach (7.19) von Seite 205 die Identität
$$\left|\frac{H(x) - g(x)}{H(x)}\right| \equiv \left|\frac{h(x) - \mathrm{P}_4(x^2)/\mathrm{Q}_4(x^2)}{h(x)}\right| \leq \varepsilon(\mathrm{app}, 2)$$

3. Im Quelltext `Approx-Error.cpp` muss daher in der C-XSC Funktion
```
l_itaylor STD(const l_itaylor& x)
```
die Funktion $h(x) := H(x)/x$ und nicht $H(x)$ definiert werden, vgl. (7.18).

4. $\displaystyle\sum_{k=0}^{14} a_k \cdot x^{2k}$ wird in (7.18) auf Seite 205 nach dem Horner-Schema berechnet; dabei gilt nach (7.15) von Seite 203
$$a_{14} = \frac{2^{14}}{1 \cdot 3 \cdot \ldots \cdot 29} = \frac{16384}{1 \cdot 3 \cdot \ldots \cdot 29}, \qquad a_k = \frac{a_{k+1} \cdot (2k+3)}{2};$$

5. Beachten Sie bitte, dass a_{14} in `STD(const l_itaylor& x)` für die Taylorarithmetik durch `t = const_l_itaylor(get_order(x),a14);` als Konstante und nicht als unabhängige Variable zu definieren ist, vgl. Seite 206.

Kapitel 8

Spezielle Standardfunktionen

In diesem Kapitel werden mehrere Beispiele zur Abschätzung von relativen Fehlern für spezielle Standardfunktionen des **C-XSC** Systems angegeben.

8.1 $\sqrt{x+1}-1$

8.1.1 Punktargumente

Als Anwendungsbeispiel betrachten wir die reelle Funktion

$$f : D_f = [-1, +\infty) \to I\!R, \quad \text{mit} \quad f(x) = \sqrt{x+1} - 1,$$

die für alle Maschinenzahlen $\widetilde{x} \in X = [-1, \texttt{MaxReal}]$ auszuwerten ist. Bezeichnet man mit $\widetilde{f}(\widetilde{x})$ den auf der Maschine berechneten Näherungswert für $f(\widetilde{x}) \approx \widetilde{f}(\widetilde{x})$, so soll ein Schranke $\Delta(f)$ des absoluten Fehlers berechnet werden, falls die Funktionswerte $f(\widetilde{x})$ im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{Minreal})$ liegen:

$$|\widetilde{f}(\widetilde{x}) - f(\widetilde{x})| \leq \Delta(f) \quad \forall\, f(\widetilde{x}) \in U$$

Für alle Funktionswerte $f(\widetilde{x}) \notin U$ aus dem normalisierten Bereich ist eine Schranke $\varepsilon(f)$ des relativen Fehlers zu berechnen:

$$\left| \frac{\widetilde{f}(\widetilde{x}) - f(\widetilde{x})}{f(\widetilde{x})} \right| \leq \varepsilon(f) \quad \forall\, f(\widetilde{x}) \notin U.$$

Die Schranken $\Delta(f)$ und $\varepsilon(f)$ werden zur Implementierung einer Intervallfunktion benötigt, mit der zu einem vorgegebenen Argumentintervall $[a, b] \subseteq X$ der entsprechende Wertebereich $W_{f,[a,b]} = \{y \in I\!R \,|\, y = f(x) \wedge x \in [a, b]\}$ von $f(x)$ garantiert eingeschlossen wird. Zur Implementierung von $f(x)$ wird das Maschinenintervall X in sieben Teilintervalle A_i unterteilt, und in jedem A_i wird $f(x)$ durch eine geeignete

Funktion $g_i(x)$ approximiert. $s = 7.99999\ldots \in S(2,53)$ wird auf Seite 216 definiert. In nachfolgender Tabelle ist für jeden der sieben Teilbereiche A_i die entsprechende Approximationsfunktion $g_i(x)$ angegeben:

Bereich A_i	Approximationsfunktion $g_i(x)$	Beziehung
$A_1 = [-1, -\frac{67}{128}]$	$g_1(x) = \sqrt{x+1} - 1$	$f(x) \equiv g_1(x)$
$A_2 = (-\frac{67}{128}, -2^{-50})$	$g_2(x) = \dfrac{x}{\sqrt{x+1}+1}$	$f(x) \equiv g_2(x)$
$A_3 = [-2^{-50}, +2^{-50}]$	$g_3(x) = \dfrac{x}{2}$	$f(x) \approx g_3(x)$
$A_4 = (2^{-50}, s]$	$g_4(x) = \dfrac{x}{\sqrt{x+1}+1}$	$f(x) \equiv g_4(x)$
$A_5 = (s, 2^{52}]$	$g_5(x) = \sqrt{x+1} - 1$	$f(x) \equiv g_5(x)$
$A_6 = (2^{52}, 2^{104})$	$g_6(x) = \sqrt{x} - 1$	$f(x) \approx g_6(x)$
$A_7 = [2^{104}, \texttt{MaxReal}]$	$g_7(x) = \sqrt{x}$	$f(x) \approx g_7(x)$

Wertet man z.B. $g_6(x)$ auf dem Rechner für eine Maschinenzahl $\widetilde{x} \in A_6 \cap S(2,53)$ aus, so gilt für das Maschinenergebnis $\widetilde{g_6}(\widetilde{x})$

$$f(\widetilde{x}) \approx \widetilde{g_6}(\widetilde{x}) = \widetilde{\sqrt{\widetilde{x}}} \ominus 1 \equiv \texttt{sqrt}(\widetilde{x}) \ominus 1$$

dabei ist $\widetilde{\sqrt{\widetilde{x}}} \equiv \texttt{sqrt}(\widetilde{x})$ die in **C-XSC** implementierte schnelle Quadratwurzel mit $\varepsilon(\texttt{sqrt}) = 2.220447 \cdot 10^{-16}$ und $\texttt{sqrt}(0) = 0$. $g_3(x), g_6(x)$ und $g_7(x)$ sind schnelle Approximationen von $f(x)$ in den angegebenen Bereichen, und es gilt die Identität:

$$f(x) = \sqrt{x+1} - 1 \equiv \frac{x}{\sqrt{x+1}+1} \quad \forall x \in D_f,$$

wobei der letzte Ausdruck rechts zur Auswertung in der Umgebung des Ursprungs besonders geeignet ist, da hier die störenden Auslöschungseffekte des ursprünglichen Funktionsterms $f(x) \equiv g_1(x)$ nicht mehr auftreten.

Wir kommen jetzt zur Fehlerabschätzung in den sieben Bereichen A_ν. Da in den Bereichen A_3, A_6 und A_7 die Funktion $f(x)$ jeweils durch $g_\nu, \nu = 3, 6, 7$ approximiert wird, ist der Approximationsfehler nach (7.1), (7.2) von Seite 186 entsprechend zu berücksichtigen.

$A_1 = [-1, -\frac{67}{128}]$

Wegen $f(x) \equiv g_1(x) = \sqrt{x+1} - 1$ gilt $\varepsilon(app) = 0$, so dass nur der relative Auswertefehler für $g_1(x)$ abzuschätzen ist. Dies erfolgt mit Hilfe des **C-XSC** Programms `sqrtp1m11.cpp`

```
// Programm:  sqrtp1m11.cpp zur Abschätzung des relativen
//            Auswertefehlers bei hochgenauer Arithmetik.
#include "bnd_util.hpp"   // Wegen Term_relh(), Term_Dat(), ...
#include <iostream>       // Wegen cout
using namespace cxsc;
using namespace std;

real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x) = sqrt(x+1)-1 für alle Maschinenzahlen x aus dem
// Teilintervall Xi.
{
    interval z,z1;
    real d,d1;
    abs_1px_delh(Xi,0,z,d);  // d: absoluter Fehler bei der Berechnung
     // von x+1, mit Maschinenzahlen x aus dem Intervall Xi; z = 1+Xi;
    abs_sqrt_abs(z,d,z1,d1); // z1: sqrt(1+x);  d1: absl. Fehler
    rel_xm1_delh(z1,d1,z,d);
    return d;  // Rückgabe des relativen Auswertefehlers
} // Term

int main()
{
// Berechnung einer relativen Fehlerschranke bez. der Auswertung
// eines Terms bei hochgenauer Arithmetik.

    interval X;  // Intervall der Maschinen-Argumente des Terms
    real bnd,diam;

    while(1) // Endlosschleife
    {
        X = interval(succ(-1),-0.5234375);
        cout << "diam = ? "; cin >> diam;
        bnd = Term_relh(Term,X,diam); // bnd: relative Fehlerschranke
        // bei nur hochgenauer Arithmetik bei der Termauswertung.
```

```
        cout << RndUp
             << "Relat. Fehlerschranke bei hochgenauer Arithmetik = "
             << bnd << endl << endl;
    }
}
```

In der Funktion `Term(...)` wird dabei der relative Fehler der Funktion $g_1(x)$ im Teilintervall $Xi \subseteq X = [\texttt{succ}(-1), -\frac{67}{128}] \subset A_1 = [-1, -\frac{67}{128}]$ abgeschätzt. Wählt man $X = A_1$, so kann der relative Fehler nicht abgeschätzt werden. Für $\widetilde{x} = -1$ gilt jedoch $\widetilde{g}_1(-1) = -1 = g_1(-1)$, so dass der relative Fehler für $\widetilde{x} = -1$ verschwindet. Das obige Programm liefert daher mit $\texttt{diam} = 10^{-5}$ das Ergebnis:

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_1(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f, A_1) := 4.950034 \cdot 10^{-16}$$

$A_2 = (-\frac{67}{128}, -2^{-50})$

Wegen $f(x) \equiv g_2(x) = x/(\sqrt{x+1}+1)$ gilt $\varepsilon(app) = 0$, so dass nur der relative Auswertefehler für $g_2(x)$ abzuschätzen ist. Dies erfolgt wieder mit Hilfe des Programms `sqrtp1m11` von Seite 213, wobei mit der Funktion `Term(..)` der relative Auswertefehler der Funktion $g_2(x)$ im Teilintervall $Xi \subseteq A_2 = X = [-\frac{67}{128}, -2^{-50}]$ abgeschätzt wird. Das obige Programm `sqrtp1m11` liefert dann mit Hilfe der Funktion `Term()` von Seite 214 das Ergebnis:

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_2(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f, A_2) := 4.718448 \cdot 10^{-16},$$

wobei X jetzt realisiert wird durch: `X = interval(-0.5234375,comp(-0.5,-49));` und die Funktion `Term(...)` gegeben ist durch:

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x) = x/(sqrt(x+1)+1) für alle Maschinenzahlen x aus dem
// Teilintervall Xi.
{
    interval z,z1;
    real d,d1;
    abs_1px_delh(Xi,0,z,d);  // d: absoluter Fehler bei der Berechnung
     // von x+1, mit Maschinenzahlen x aus dem Intervall Xi; z = 1+Xi;
    abs_sqrt_abs(z,d,z1,d1);  // z1: sqrt(1+Xi);  d1: absl. Fehler
    abs_1px_delh(z1,d1,z,d);  // z : sqrt(1+Xi)+1; d: absl. Fehler
    rel_divh1(Xi,0,z,d,z1,d1);
    return d1;  // Rückgabe des relativen Auswertefehlers
} // Term
```

$A_3 = [-2^{-50}, +2^{-50}]$

In diesem Intervall müssen wir das Teilintervall $A_{3,0} := [-2^{-1021}, +2^{-1021}] \subset A_3$ gesondert betrachten, da für alle Maschinenzahlen $\widetilde{x} \in A_{3,0} \cap S(2,53)$ die zugehörigen Maschinenwerte $\widetilde{g}_3(\widetilde{x})$ in folgendem Intervall $[-\texttt{MinReal}, +\texttt{MinReal}]$ liegen:

$$|\widetilde{x}| \le 2^{-1021} \implies |\widetilde{g}_3(\widetilde{x})| \equiv |\widetilde{x} \oslash 2| \le \texttt{MinReal} = 2^{-1022} \tag{8.1}$$

$$|\widetilde{x}| > 2^{-1021} \implies |\widetilde{g}_3(\widetilde{x})| \equiv |\widetilde{x} \oslash 2| \ge \texttt{MinReal} = 2^{-1022} \tag{8.2}$$

Der Beweis von (8.1) und (8.2) ergibt sich unmittelbar aus der Forderung, dass die benutzte Arithmetik "faithful" sein soll, d.h. sind u, v zwei beliebige Maschinenzahlen, $\circ \in \{+, -, *, /\}$, $\bullet \in \{\oplus, \ominus, \odot, \oslash\}$, $v \neq 0$, falls $\circ = /$ und $w := u \circ v \in [s_1, s_2)$, wobei s_1, s_2 zwei aufeinanderfolgende Maschinenzahlen sind, so soll für die Rundung von w in das Gleitpunktsystem $S(2,53)$ gelten:

$$\begin{aligned} w = s_1 &\implies u \bullet v = s_1 \\ w \neq s_1 &\implies u \bullet v = s_1 \ \vee \ u \bullet v = s_2, \end{aligned}$$

wobei in der zweiten Zeile s_1 bzw. s_2 durch den jeweils eingestellten Rundungsmodus bestimmt wird. Eine Arithmetik ist übrigens immer dann "faithful", wenn sie dem IEEE–Standard genügt, was stets vorausgesetzt wird.

Da für $|\widetilde{x}| \le 2^{-1021}$, d.h. für $\widetilde{x} \in A_{3,0}$ der relative Fehler mit $\widetilde{x} \to 0$ viel zu groß wird, ist es sinnvoll, in diesem Bereich nur eine Schranke des absoluten Fehlers zu berechnen:

$$|f(\widetilde{x}) - \widetilde{g}_3(\widetilde{x})| = |f(\widetilde{x}) - \widetilde{x} \oslash 2| \le \Delta(f) = ? \quad \forall\, \widetilde{x} \in A_{3,0} \cap S(2,53)$$

Bevor wir nach (7.2) $\Delta(f)$ berechnen, muss eine Schranke $\Delta(app)$ des Fehlers bzgl. der Approximation $f(x) = \sqrt{x+1} - 1 \approx x/2$ bestimmt werden:

$$\begin{aligned} \Delta_{app} &= \left|\frac{x}{2} + 1 - \sqrt{x+1}\right| = \left|\frac{x}{2} - \frac{x}{\sqrt{x+1}+1}\right| = \frac{|x|}{2} \cdot \left|\frac{\sqrt{x+1}-1}{1+\sqrt{x+1}}\right| \\ &\le \frac{|x|}{2} \cdot \frac{\sqrt{x+1}-1}{1} = \frac{|x|}{2} \cdot \left|\frac{x}{\sqrt{x+1}+1}\right| \le \frac{x^2}{2} \end{aligned}$$

Für alle $x \in A_{3,0} = [-2^{-1021}, +2^{-1021}]$ gilt damit:

$$\Delta_{app} \le \frac{2^{-2\cdot 1021}}{2} = 2^{-2041} = \Delta(app)$$

Nach (7.2) benötigen wir jetzt noch $\Delta(g)$. Wegen $\widetilde{x}/2 \in [-\texttt{MinReal}, +\texttt{MinReal}]$ gilt die Abschätzung[1] $|\widetilde{x}/2 - \widetilde{x} \oslash 2| < \texttt{minreal} = 2^{-1074} = \Delta(g)$. Nach (7.2) folgt damit

[1] Im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{MinReal})$ haben die denormalisierten Zahlen alle den gleichen Abstand $\texttt{minreal} = 2^{-1074}$.

für alle $\widetilde{x} \in A_{3,0} = [-2^{-1021}, +2^{-1021}]$:

$$|f(\widetilde{x}) - \widetilde{g}_3(\widetilde{x})| \leq 2^{-2041} + 2^{-1074} < 2 \cdot \texttt{minreal} = 2^{-1073} = \Delta(f). \tag{8.3}$$

Für die Abschätzung des relativen Fehlers im restlichen Bereich $A_3 \backslash A_{3,0}$ benötigen wir nach (7.1) neben einer Schranke $\varepsilon(g)$ des relativen Auswertefehler zunächst eine Schranke $\varepsilon(app)$ des relativen Approximationsfehlers:

$$\begin{aligned} |\varepsilon_{app}| &= \left| \frac{(\sqrt{x+1} - 1 - \frac{x}{2})(\sqrt{x+1}+1)}{x} \right| = \left| \frac{x - \frac{x}{2}(\sqrt{x+1}+1)}{x} \right| \\ &= \left| 1 - \frac{1}{2} \cdot \left(\sqrt{x+1}+1\right) \right| = \frac{1}{2} \cdot \left|\sqrt{x+1} - 1\right| = \frac{|x|/2}{\sqrt{x+1}+1} \leq \frac{|x|}{2} \end{aligned}$$

Wegen $|x| \leq 2^{-50}$ folgt damit für die Schranke $\varepsilon(app)$ des relativen Approximationsfehlers:

$$|\varepsilon_{app}| \leq 2^{-51} = \varepsilon(app) = 4.44089\ldots \cdot 10^{-16} \quad \forall x \in A_3 \backslash A_{3,0}.$$

Nach (8.2) liegen alle Maschinenergebnisse $\widetilde{x} \oslash 2$ im normalisierten Bereich, so dass die Division durch 2 auf der Maschine rundungsfehlerfrei erfolgt, d.h. es gilt $\varepsilon(g) = 0$. Damit erhalten wir nach (7.1) im Restbereich $A_3 \backslash A_{3,0}$ für den relativen Fehler die Abschätzung:

$$\left| \frac{f(\widetilde{x}) - \widetilde{g}_3(\widetilde{x})}{f(\widetilde{x})} \right| \leq \varepsilon(f, A_3 \backslash A_{3,0}) = \varepsilon(app) = 2^{-51} \quad \forall\, \widetilde{x} \in A_3 \backslash A_{3,0}.$$

$A_4 = (+2^{-50}, s]$;

Für die Maschinenzahl $s \in S(2, 53)$ gilt:

$$\begin{aligned} s &= 9007199254740984.0/1125899906842624.0 \\ &= 7.99999999999999289457264239899814128875732421875 \end{aligned}$$

Wegen $f(x) \equiv g_4(x) = x/(\sqrt{x+1}+1)$ gilt $\varepsilon(app) = 0$, so dass nur der relative Auswertefehler für $g_4(x)$ abzuschätzen ist. Dies erfolgt wieder mit Hilfe des Programms `sqrtp1m12.cpp`

```
// Programm:  sqrtp1m12.cpp zur Abschätzung des relativen
//            Auswertefehlers bei hochgenauer Arithmetik.

#include "bnd_util.hpp"   // Wegen Term_relh()
#include <iostream>       // Wegen cout

using namespace cxsc;
using namespace std;
```

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x) = x/(sqrt(x+1)+1) für alle Maschinenzahlen x aus dem
// Teilintervall Xi.
{
    interval z,z1;
    real d,d1;
    abs_1px_delh(Xi,0,z,d);  // d: absoluter Fehler bei der Berechnung
     // von x+1, mit Maschinenzahlen x aus dem Intervall Xi; z = 1+Xi;
    abs_sqrt_abs(z,d,z1,d1);  // z1: sqrt(1+Xi);  d1: absl. Fehler
    abs_1px_delh(z1,d1,z,d);  // z : sqrt(1+Xi)+1; d: absl. Fehler
    rel_divh1(Xi,0,z,d,z1,d1);
    return d1;  // Rückgabe des relativen Auswertefehlers
} // Term

int main()
{
    interval X;  // Intervall der Maschinen-Argumente des Terms
    real bnd,diam,s;
    s = 9007199254740984.0/1125899906842624.0;
    // s = 7.99999999999999289457264239899814128875732421875;
    // diam: Mantissendurchmesser der Teilintervalle (10^(-5))
    while(1) // Endlosschleife
    {
        X = interval(comp(0.5,-49),s);
        cout << "diam = ? "; cin >> diam;
        bnd = Term_relh(Term,X,diam); // bnd: relative Fehlerschranke
        // bei nur hochgenauer Arithmetik bei der Termauswertung.
        cout << RndUp
             << "Relat. Fehlerschranke bei hochgenauer Arithmetik = "
             << bnd << endl << endl;
    }
}
```

Mit der Funktion `Term(..)` wird dabei der relative Auswertefehler von $g_4(x)$ im Teilintervall $Xi \subseteq A_4 = X = [\texttt{comp}(0.5,-49), s]$ abgeschätzt. Das obige Programm `sqrtp1m12` liefert dann mit dem Eingabewert `diam` $= 10^{-5}$ das Ergebnis:

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_4(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f, A_4) := 4.996004 \cdot 10^{-16}.$$

$A_5 = (s, 2^{52}]$;

Für die Maschinenzahl $s \in S(2,53)$ gilt:

$$\begin{aligned} s &= 9007199254740984.0/1125899906842624.0 \\ &= 7.99999999999999289457264239899814128875732421875 \end{aligned}$$

Wegen $f(x) \equiv g_5(x) = \sqrt{x+1} - 1$ gilt $\varepsilon(app) = 0$, so dass nur der relative Auswertefehler für $g_5(x)$ abzuschätzen ist. Dies erfolgt wieder mit Hilfe des Programms `Term_relh` von Seite 216, wobei die Funktion `Term()` wie auf Seite 213 zu definieren ist und das Intervall `X` durch die Anweisung `X = interval(s,comp(0.5,53))` realisiert werden muss. Das so abgeänderte Programm `sqrtp1m12` liefert dann mit dem Eingabewert `diam` $= 10^{-5}$ das Ergebnis:

$$\left| \frac{f(\widetilde{x}) - \widetilde{g}_5(\widetilde{x})}{f(\widetilde{x})} \right| \leq \varepsilon(f, A_5) := 4.070830 \cdot 10^{-16}.$$

$A_6 = (2^{52}, 2^{104})$;

Um den relativen Fehler für alle $\widetilde{x} \in A_6 \cap S(2,53)$ nach (7.1) abschätzen zu können, muss zunächst eine Schranke $\varepsilon(app)$ des relativen Approximationsfehlers bez. $f(x) \approx g_6(x) = \sqrt{x} - 1$ berechnet werden. Für $x \in A_6$ gilt:

$$\begin{aligned} |\varepsilon_{app}| &= \frac{\sqrt{x+1} - 1 - (\sqrt{x} - 1)}{\sqrt{x+1} - 1} = \frac{\sqrt{x+1} - \sqrt{x}}{\sqrt{x+1} - 1} = \frac{(\sqrt{x+1} - \sqrt{x})(\sqrt{x+1} + 1)}{x} \\ &= \frac{(\sqrt{x+1} - \sqrt{x})(\sqrt{x+1} + \sqrt{x})(\sqrt{x+1} + 1)}{x(\sqrt{x+1} + \sqrt{x})} = \frac{\sqrt{x+1} + 1}{x(\sqrt{x+1} + \sqrt{x})} \leq \frac{1}{x} \end{aligned}$$

Für $x > 2^{52}$ folgt damit: $|\varepsilon_{app}| \leq 2^{-52} < 2.220447 \cdot 10^{-16} = \varepsilon(app)$. Die Berechnung einer Schranke $\varepsilon(g_6)$ des relativen Auswertefehlers erfolgt wieder mit dem Programm `sqrtp1m12` von Seite 216, wobei die Funktion `Term()` jetzt für $g_6(x) = \sqrt{x} - 1$ zu ersetzen ist durch:

```
real Term(const interval& Xi)
// Berechnet den relativen Auswertefehler des Terms
// T(x) = sqrt(x)-1 für alle Maschinenzahlen x aus dem
// Teilintervall Xi.
{
    interval z,z1;
    real d,d1;
    abs_sqrt_abs(Xi,0,z1,d1); // z1: sqrt(x),x aus Xi; d1: abs. Fehler
    rel_xm1_delh(z1,d1,z,d);
```

```
    return d;  // Rückgabe des relativen Auswertefehlers
}
```

und `X` $\supset A_6$ zu realisieren ist durch `X = interval(comp(0.5,53),comp(0.5,105));` Mit dem Eingabewert `diam` $= 10^{-5}$ liefert das Programm `sqrtp1m12` das Ergebnis:

$$\left|\frac{g_6(\widetilde{x})-\widetilde{g}_6(\widetilde{x})}{g_6(\widetilde{x})}\right| \leq \varepsilon(g_6) := 2.220469 \cdot 10^{-16}.$$

Nach (7.1) können wir jetzt mit den berechneten Werten $\varepsilon(app)$ und $\varepsilon(g_6)$ eine Schranke für den relativen Fehler direkt angeben:

$$\left|\frac{f(\widetilde{x})-\widetilde{g}_6(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f, A_6) := 4.441139 \cdot 10^{-16}.$$

A_7 = [2^{104}, `MaxReal`];

In diesem Intervall wird $f(x)$ approximiert durch $g_7(x) = \sqrt{x}$, und da der relative Auswertefehler von $g_7(x)$ mit $\varepsilon(g_7) = 2.220447 \cdot 10^{-16}$ schon bekannt ist, muss nur noch eine Schranke $\varepsilon(app)$ des relativen Approximationsfehlers berechnet werden:

$$\begin{aligned}
|\varepsilon_{app}| &= \frac{\sqrt{x}+1-\sqrt{x+1}}{\sqrt{x+1}-1} = \frac{(\sqrt{x+1}+1)(\sqrt{x}+1-\sqrt{x+1})}{x} \\
&= \frac{\sqrt{x}\cdot(\sqrt{x+1}+1)-x}{x} = \frac{\sqrt{x+1}+1-\sqrt{x}}{\sqrt{x}} \\
&= \frac{\{\sqrt{x+1}+(1-\sqrt{x})\}\cdot\{\sqrt{x+1}-(1-\sqrt{x})\}}{\sqrt{x}\cdot\{\sqrt{x+1}-(1-\sqrt{x})\}} \\
&= \frac{x+1-(1-2\sqrt{x}+x)}{\sqrt{x}\cdot\{\sqrt{x+1}+\sqrt{x}-1\}} = \frac{2\sqrt{x}}{\sqrt{x}\cdot\{\sqrt{x+1}+\sqrt{x}-1\}} \\
&= \frac{2}{\sqrt{x+1}+\sqrt{x}-1} \leq \frac{2}{2\sqrt{x}-1}
\end{aligned}$$

Für alle $x \in A_7$ gilt $x \geq 2^{104}$, und wirklich einfache Rechnungen zeigen, dass dies äquivalent ist zu

$$\frac{2}{2\sqrt{x}-1} \leq \frac{2}{2^{53}-1} \quad \Longleftrightarrow \quad x \geq 2^{104}.$$

Eine Schranke $\varepsilon(app)$ des relativen Approximationsfehlers ist damit gegeben durch:

$$|\varepsilon_{app}| \leq \frac{2}{2\sqrt{x}-1} \leq \frac{2}{2^{53}-1} < 2.220447 \cdot 10^{-16} = \varepsilon(app) \quad \forall x \in A_7.$$

Nach (7.1) können wir jetzt mit den berechneten Werten $\varepsilon(app)$ und $\varepsilon(g_7)$ eine Schranke für den relativen Fehler direkt angeben:

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_7(\widetilde{x})}{f(\widetilde{x})}\right| \leq \varepsilon(f, A_7) := 4.440895 \cdot 10^{-16}.$$

Zusammenfassung:

$$\begin{aligned} |f(\widetilde{x}) - \widetilde{g}_3(\widetilde{x})| &< \Delta(f) = 2 \cdot \texttt{minreal} = 2^{-1073} \quad \forall\, \widetilde{x} \in A_{3,0} = [-2^{-1021}, +2^{-1021}] \\ \left|\frac{f(\widetilde{x}) - \widetilde{g}_\nu(\widetilde{x})}{f(\widetilde{x})}\right| &< \varepsilon(f) = 4.996004 \cdot 10^{-16} \quad \forall\, \widetilde{x} \in [-1, +\texttt{MaxReal}] \setminus A_{3,0} \end{aligned}$$

$\varepsilon(f)$ ist dabei das Maximum aller relativen Fehlerschranken in den auf Seite 212 angegebenen sieben Teilintervallen $A_\nu,\ \nu = 1, 2, \ldots, 7$.

Die Fehlerschranken $\Delta(f), \varepsilon(f)$ spielen bei der Implementierung einer Intervallfunktion eine entscheidende Rolle. Für deren Realisierung benötigt man natürlich auch die Punktfunktion `sqrtp1m1()`, mit der die fehlerbehafteten Funktionswerte $\widetilde{f}(\widetilde{x}) = \widetilde{g}_\nu(\widetilde{x})$ vorher zu berechnen sind. Der Quelltext von `sqrtp1m1()` ist in dem nachfolgenden Testprogramm angegeben:

```
// program sqrtp1m13.cpp calculating an approximation of sqrt(x+1)-1;

#include <iostream>   // for cout
#include <rmath.hpp>  // for sqrt()

using namespace cxsc;
using namespace std;

const real Sqrtp1m1_s = 9007199254740984.0 / 1125899906842624.0;

real Sqrtp1m1(const real& x) throw()
// Sqrtp1m1(x) = sqrt(x+1)-1;
{
    real y = x;   int ex = expo(x);
    if (ex<=-50) times2pown(y,-1);  // |x|<2^(-50); fast division by 2
    else if (ex>=105) y = sqrt(x);  // x >= 2^(+104) = 2.02824...e+31
    else if (ex>=53) y = sqrt(x)-1; // x >= 2^(+52)  = 4.50359...e+15
    else if (x>-0.5234375 && x<=Sqrtp1m1_s ) y = x / (sqrt(x+1) + 1);
    else y = sqrt(x+1)-1;
    return y;
}
```

```
int main()
{
    real x;

    while (1)
    {
        cout << "x = ? ";  cin >> RndNext >> x;
        cout << SetPrecision(16,16) << Scientific
             << "sqrtp1m1(x) = " << Sqrtp1m1(x) << endl << endl;
    }
} // end of main
```

Ergebnisse: (In der **C-XSC** Bibliothek wird `Sqrtp1m1` mit `sqrtp1m1` bezeichnet!)

$x = 3$ liefert den exakten Funktionswert: `sqrtp1m1(x) = 1.0000000000000000`
$x = 9.5367431640625 \cdot 10^{-7}$ liefert: `sqrtp1m1(x) = 4.7683704451634149E-007`.
Beachten Sie bitte, dass das letzte Funktionsargument x den Wert 2^{-20} besitzt und damit rundungsfehlerfrei im `real`-Format gespeichert werden kann. Bei der Dateneingabe treten hier also keine Konversionsfehler auf.
Beachten Sie ferner, dass die naive Auswertung des Ausdrucks $\sqrt{x+1}-1$ mit dem gleichen Argument $x = 2^{-20}$ den Wert `4.7683704451628728E-007` liefert, der sich wegen der verstärkten Auslöschung vom obigen Funktionswert schon deutlich unterscheidet. Der mit Maple7.0 zum gleichen Argument $x = 2^{-20}$ berechnete exakte Funktionswert lautet: $\sqrt{2^{-20}+1}-1 = 4.76837044516341488460282814... \cdot 10^{-7}$.

8.1.2 Intervallargumente

Mit Hilfe der auf Seite 220 angegebenen absoluten und relativen Fehlerschranken soll jetzt noch die Funktion `sqrtp1m1()` für Intervallargumente $\mathtt{x} \in IR$ realisiert werden. Der Funktionsaufruf `y = sqrtp1m1(x)` liefert dann mit dem Intervall $\mathtt{y} \in IR$ eine garantierte Einschließung aller reellen Funktionswerte $y = \sqrt{1+x}-1$, mit $x \in \mathtt{x} \cap \mathbb{R}$.

Für ein beliebiges Maschinenargument $\widetilde{x} \in \mathtt{x}$ berechnen wir zunächst eine Unterschranke des entsprechenden Funktionswertes $\sqrt{1+\widetilde{x}}-1$. Für $\widetilde{x}=0$ gilt offensichtlich:

$$\widetilde{x} = 0 \quad \Longrightarrow \quad 0 \leq \sqrt{1+\widetilde{x}} - 1 \tag{8.4}$$

und nach Seite 220 erhält man dann im gelochten Bereich $0 < |\widetilde{x}| \leq 2^{-1021}$ für die exakten Funktionswerte $\sqrt{\widetilde{x}+1}-1$ die garantierte Unterschranke:

$$0 < |\widetilde{x}| \leq 2^{-1021} \quad \Longrightarrow \quad \mathtt{sqrtp1m1}(\widetilde{x}) - \Delta(f) \leq \sqrt{\widetilde{x}+1} - 1 \tag{8.5}$$

Im restlichen Definitionsbereich $|\widetilde{x}| > 2^{-1021} \wedge \widetilde{x} \geq -1$ liegen jetzt nur noch normalisierte Zahlen der Form $\widetilde{x} = m \cdot 2^{ex}$, mit $1 \leq |m| < 2$ und $ex \geq -1021$.

Wegen $\sqrt{x+1}-1 \leq x/2$ für alle $x \geq -1$ kann man im restlichen Definitionsbereich zu gegebenem Funktionsargument $\widetilde{x}$ mit $\mathtt{pred}(\widetilde{x}/2)$ günstige, d.h. möglichst große Unterschranken zum Funktionswert $\sqrt{\widetilde{x}+1}-1$ berechnen, wenn man nicht zu große $|\widetilde{x}|$-Werte wählt. Wir formulieren diese Idee im folgenden Satz:

$$2^{-1021} \leq |\widetilde{x}| < (\sqrt{2} - 2^{-52}) \cdot 2^{-51} \quad \Longrightarrow \quad \mathtt{pred}\left(\frac{\widetilde{x}}{2}\right) \leq \sqrt{\widetilde{x}+1} - 1 \tag{8.6}$$

Dabei wird die Einschränkung $2^{-1021} \leq |\widetilde{x}|$ hier nur getroffen, da im Fall $|\widetilde{x}| < 2^{-1021}$ auf der Maschine gilt: $\widetilde{x}/2 \neq \widetilde{x} \oslash 2$ und damit $\mathtt{pred}(\widetilde{x}/2)$ nicht mehr korrekt berechnet werden kann. Außerdem haben wir in (8.4) und (8.5)) bereits Unterschranken für $|\widetilde{x}| \leq 2^{-1021}$ angegeben.

Zum Beweis von (8.6) schreiben wir nach (1.7) von Seite 8:

$$\mathtt{pred}\left(\frac{\widetilde{x}}{2}\right) = \begin{cases} \frac{\widetilde{x}}{2} - 2^{-53+ex-1} & \text{falls } m = 1 \\ \frac{\widetilde{x}}{2} - 2^{-52+ex-1} & \text{sonst} \end{cases} = \begin{cases} \frac{\widetilde{x}}{2} - 2^{ex-54} & \text{falls } m = 1 \\ \frac{\widetilde{x}}{2} - 2^{ex-53} & \text{sonst,} \end{cases}$$

und (8.6) ist bewiesen, wenn wir gezeigt haben:

$$\begin{aligned} & 1 + \frac{\widetilde{x}}{2} - 2^{ex-53} \leq \sqrt{\widetilde{x}+1} \\ \Longleftrightarrow \quad & 1 + m \cdot 2^{ex-1} - 2^{ex-53} \leq \sqrt{\widetilde{x}+1} \\ \Longleftrightarrow \quad & 1 + 2^{ex-1}(m - 2^{-52}) \leq \sqrt{\widetilde{x}+1} \\ \Longleftrightarrow \quad & -2^{ex-52} + 2^{2ex-2}(m-2^{-52})^2 \leq 0 \\ \Longleftrightarrow \quad & 2^{ex+50}(m-2^{-52})^2 \leq 1 \end{aligned} \tag{8.7}$$

Zum Beweis von (8.7) setzen wir nach (8.6) zunächst voraus:

$$ex = -51 \quad \text{und} \quad 1 \le |m| < \sqrt{2} - 2^{-52},$$

und zu zeigen bleibt dann nach (8.7): $\qquad |m - 2^{-52}| \le \sqrt{2}.$

1. Sei $m \ge 2^{-52}$, dann ist obige Ungleichung äquivalent mit $m \le \sqrt{2} + 2^{-52}$, und dies ist erfüllt, da nach Voraussetzung für positives m gilt: $m < \sqrt{2} - 2^{-52}$.
2. Sei $m < 2^{-52}$, dann ist obige Ungleichung äquivalent mit $\sqrt{2} \ge -m + 2^{-52}$, d.h. $m \ge -\sqrt{2} + 2^{-52}$, und dies ist erfüllt, da nach Voraussetzung für negatives m gilt: $m > -\sqrt{2} + 2^{-52}$.

Zum Beweis von (8.7) setzen wir nach (8.6) jetzt noch voraus:

$$ex \le -52 \quad \text{und} \quad 1 \le |m| < 2,$$

und zu zeigen bleibt dann nach (8.7):

$$2^{-50} \ge 2^{ex} \cdot (m - 2^{-52})^2 \tag{8.8}$$

Für $1 \le |m| < 2$ gilt: $1 \le |m| \le 2 - 2^{-52}$ und damit:

$$2^{ex} \cdot (m - 2^{-52})^2 \le 2^{ex} \cdot (-2 + 2^{-52} - 2^{-52})^2 = 2^{ex+2}$$

(8.8) ist damit erfüllt, wenn gilt: $2^{-50} \ge 2^{ex+2}$, d.h. wenn $ex \le -52$, und dies ist gerade die obige Voraussetzung ∎

Im restlichen Definitionsbereich $\widetilde{x} \ge -1 \wedge |\widetilde{x}| \ge 2^{-51} \cdot (\sqrt{2} - 2^{-52})$ wird zu vorgegebenem Maschinenargument $\widetilde{x}$ eine garantierte Unterschranke des Funktionswertes $\sqrt{\widetilde{x}+1} - 1$ mit Hilfe des relativen Fehlers $\varepsilon(f) = 4.996004 \cdot 10^{-16}$ berechnet. Mit den im `real`-Format exakt darstellbaren Quotienten[2]

$$\begin{aligned} \texttt{q\_sqrtp1m1p} &= 4503599627370502.0/4503599627370496.0\,, \quad (1 + \texttt{eps}) \\ \texttt{q\_sqrtp1m1m} &= 9007199254740984.0/9007199254740992.0\,, \quad (1 - \texttt{eps}) \end{aligned}$$

lauten diese Unterschranken:

$$\begin{aligned} \texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1m} &\le \sqrt{\widetilde{x}+1} - 1, \text{ falls } \widetilde{x} \ge 2^{-51} \cdot (\sqrt{2} - 2^{-52}) \\ \texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1p} &\le \sqrt{\widetilde{x}+1} - 1, \text{ falls } -1 \le \widetilde{x} \le -2^{-51} \cdot (\sqrt{2} - 2^{-52}) \end{aligned}$$

Damit haben wir alle notwendigen Unterschranken berechnet, die auf nachfolgender Seite nochmals zusammengestellt sind.

[2] Zähler und Nenner dieser Quotienten erhält man wie auf Seite 80 durch Aufruf der Funktion `eps2fractions()`, die vom Modul `bnd_util` exportiert wird.

Zu vorgegebenem Funktionsargument $\widetilde{x} \geq -1$ lautet die garantierte Unterschranke des exakten Funktionswertes $\sqrt{\widetilde{x}+1}-1$:

$$0 \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } \widetilde{x}=0$$

$$\texttt{sqrtp1m1}(\widetilde{x}) - \Delta(f) \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } 0<|\widetilde{x}|\leq 2^{-1021}$$

$$\texttt{pred}\left(\frac{\widetilde{x}}{2}\right) \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } 2^{-1021}\leq|\widetilde{x}|<2^{-51}\cdot(\sqrt{2}-2^{-52})$$

$$\texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1m} \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } \widetilde{x}\geq 2^{-51}\cdot(\sqrt{2}-2^{-52})$$

$$\texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1p} \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } -1\leq\widetilde{x}\leq -2^{-51}\cdot(\sqrt{2}-2^{-52})$$

Die Unterschranken der beiden letzten Ungleichungen lassen sich bei betragsmäßig hinreichend großen Argumenten $\widetilde{x}$ noch deutlich verbessern, d.h. vergrößern, wenn man die Funktionsauswertung mit dem Punktintervall $\texttt{interval}(\widetilde{x})$ in naiver Intervallrechnung durchführt und wegen der monoton wachsenden Funktion als Unterschranke das Infimum der berechneten Einschließung wählt:

$$\texttt{Inf}(\texttt{sqrt}(\texttt{interval}(\widetilde{x})+1)-1) \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } \widetilde{x}>0.67$$

$$\texttt{Inf}(\texttt{sqrt}(\texttt{interval}(\widetilde{x})+1)-1) \leq \sqrt{\widetilde{x}+1}-1, \text{ falls } -1\leq\widetilde{x}<-0.25$$

Zu vorgegebenem Funktionsargument $\widetilde{x} \geq -1$ müssen wir jetzt noch eine garantierte Oberschranke des exakten Funktionswertes $\sqrt{\widetilde{x}+1}-1$ berechnen. Für $\widetilde{x}=0$ gilt offensichtlich:

$$\widetilde{x}=0 \quad\Longrightarrow\quad \sqrt{1+\widetilde{x}}-1\leq 0 \tag{8.9}$$

und nach Seite 220 erhält man dann im gelochten Bereich $0<|\widetilde{x}|\leq 2^{-1021}$ für die exakten Funktionswerte $\sqrt{\widetilde{x}+1}-1$ die garantierte Oberschranke:

$$0<|\widetilde{x}|\leq 2^{-1021} \quad\Longrightarrow\quad \sqrt{\widetilde{x}+1}-1 \leq \texttt{sqrtp1m1}(\widetilde{x})+\Delta(f) \tag{8.10}$$

Wegen $\sqrt{1+\widetilde{x}}-1\leq\widetilde{x}/2$ ist für betragsmäßig hinreichend kleine Argumente $\widetilde{x}$ der Wert $\widetilde{x}/2$ eine geeignete Oberschranke oder die Multiplikation von $\texttt{sqrtp1m1}(\widetilde{x})$ mit den auf Seite 223 angegebenen Quotienten `q_sqrtp1m1p` bzw. `q_sqrtp1m1m` liefert eine garantierte Oberschranke:

$$\sqrt{\widetilde{x}+1}-1 \leq \begin{cases} \dfrac{\widetilde{x}}{2} & \text{falls } |\widetilde{x}|<2^{-47} \\ \texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1p} & \text{falls } \widetilde{x}\geq 2^{-47} \\ \texttt{sqrtp1m1}(\widetilde{x}) \odot \texttt{q\_sqrtp1m1m} & \text{falls } -1\leq\widetilde{x}\leq -2^{-47} \end{cases}$$

Auch jetzt lassen sich die beiden letzten Oberschranken weiter verkleinern, wenn man für betragsmäßig hinreichend große Argumente $\widetilde{x}$ die Funktionswerte intervallmäßig auswertet und für die Oberschranke das Supremum der berechneten Einschließung wählt. Weitere Einzelheiten kann man dem nachfolgenden Quelltext der Intervallfunktion entnehmen.

```
//*************************************************************************
    const real Delta_f = 2*minreal;
    const real q_sqrtp1m1m = 9007199254740984.0 / 9007199254740992.0;
    const real q_sqrtp1m1p = 4503599627370502.0 / 4503599627370496.0;
//*************************************************************************
interval sqrtp1m1(const interval& x) throw()
// interval(a,b) is an inclusion of sqrt(x+1)-1;
// Exported by imath.hpp;      Blomquist, 05.08.03;
{
    real a=0,b=0,ix=Inf(x),sx=Sup(x);
    int ex_ix,ex_sx,sgn_ix,sgn_sx;
    // Calculating the lower bound:
    ex_ix = expo(ix);  sgn_ix = sign(ix);
    if (ex_ix<=-1021) //  ==> |ix| < 2^(-1021)
        { if (sgn_ix) a = sqrtp1m1(ix) - Delta_f; }
    else // |ix| >= 2^(-1021)
        if (ex_ix<=-51)
        {
            times2pown(ix,-1); // exact division by 2
            a = pred(ix);
        }
        else
            if (sgn_ix>0) a = (ix>0.67) ?
             Inf( sqrt(interval(ix)+1)-1 ) : sqrtp1m1(ix)*q_sqrtp1m1m;
            else a = (ix<-0.25) ?
             Inf( sqrt(interval(ix)+1)-1 ) : sqrtp1m1(ix)*q_sqrtp1m1p;
    // Calculating the upper bound:
    if (ix == sx) { ex_sx = ex_ix;  sgn_sx = sgn_ix; }
    else { ex_sx = expo(sx);  sgn_sx = sign(sx); }
    if (ex_sx<=-1021) //  ==> |sx| < 2^(-1021)
        { if (sgn_sx) b = sqrtp1m1(sx) + Delta_f; }
    else // |sx| >= 2^(-1021)
        if (ex_sx<=-47) { b = sx; times2pown(b,-1); }
        else
            if (sgn_sx>0) b = (sx>0.58) ?
             Sup( sqrt(interval(sx)+1)-1 ) : sqrtp1m1(sx)*q_sqrtp1m1p;
            else b = (sx<-0.32) ?
             Sup( sqrt(interval(sx)+1)-1 ) : sqrtp1m1(sx)*q_sqrtp1m1m;
    return interval(a,b);
} // sqrtp1m1()
```

Auch bei der Berechnung garantierter Oberschranken wird in den Fällen $\widetilde{x} > 0.58$ und $-1 \leq \widetilde{x} < -0.32$ der Funktionswert $\sqrt{\widetilde{x}+1}-1$ zunächst durch naive Intervallrechnung eingeschlossen, und die Oberschranke erhält man dann wegen der Monotonie durch die Berechnung des Supremums. Nach diesem Verfahren erhält man bessere Einschließungen als durch die Multiplikation des Maschinenwertes $\texttt{sqrtp1m1}(\widetilde{x})$ mit den Quotienten `q_sqrtp1m1p` bzw. `q_sqrtp1m1m`.

Das Beispiel zeigt, dass die Implementierung einer Intervallfunktion mit möglichst engen Einschließungen keine triviale Aufgabe ist, sondern dass in Teilbereichen der Definitionsmenge eventuell neue Abschätzungen erforderlich werden. Dabei muss man i.a. einen Kompromiss schließen zwischen der Güte der Einschließung und dem damit verbundenen Rechenaufwand.

Ergebnisse:

1. Mit der Maschinenzahl $\widetilde{x} = 2^{-20} = 9.5367431640625 \cdot 10^{-7}$ erhält man für den Funktionswert $\sqrt{\widetilde{x}+1}-1$ die Einschließung:

   $$\sqrt{\widetilde{x}+1}-1 \in [4.7683704451634106 \cdot 10^{-7}, 4.7683704451634213 \cdot 10^{-7}]$$

   Vergleichen Sie dazu auch die entsprechenden Ergebnisse von Seite 221.

2. Mit der Maschinenzahl $\widetilde{x} = 2^{+30}-1 = 1073741823$ erhält man für den exakten Funktionswert $\sqrt{\widetilde{x}+1}-1 = 32767$ die Einschließung:

   $$\sqrt{\widetilde{x}+1}-1 \in [3.2766999999999996 \cdot 10^{+4}, 3.2767000000000008 \cdot 10^{+4}]$$

3. Mit der Maschinenzahl $\widetilde{x} = \texttt{MinReal} = 2^{-1022}$ erhält man für den Funktionswert $\sqrt{\widetilde{x}+1}-1 = 1$ die Einschließung:

   $$\sqrt{\widetilde{x}+1}-1 \in [1.1125369292535997 \cdot 10^{-308}, 1.1125369292536017 \cdot 10^{-308}]$$

   Mit Maple berechnet man den exakten Funktionswert zu:

   $$\sqrt{\widetilde{x}+1}-1 = 1.11253692925360069154511\ldots \cdot 10^{-308}$$

4. Mit der Maschinenzahl $\widetilde{x} = \texttt{minreal} = 2^{-1074}$ erhält man für den Funktionswert $\sqrt{\widetilde{x}+1}-1 = 1$ die grobe Einschließung:

   $$\sqrt{\widetilde{x}+1}-1 \in [-9.8813129168249309 \cdot 10^{-324}, 9.8813129168249309 \cdot 10^{-324}]$$

   Mit Maple berechnet man den exakten Funktionswert zu:

   $$\sqrt{\widetilde{x}+1}-1 = 2.47032822920623272088284\ldots \cdot 10^{-324}$$

   Die obige grobe aber korrekte Einschließung ergibt sich nach Seite 220 aus der Addition von $\pm\Delta(f) = \pm 2 \cdot \texttt{minreal} = \pm 9.8813\ldots \cdot 10^{-324}$ zum Maschinenwert $\texttt{sqrtp1m1}(\texttt{minreal}) = 0$, wobei dieser Maschinenwert hier sogar maximal genau berechnet wurde!

8.2 $\ln(\sqrt{x^2+y^2})$

Als weiteres Beispiel zur Fehlerabschätzung betrachten wir die reelle Funktion

$$f : D_f = I\!R^2 \backslash \{(0,0)\} \to I\!R, \quad \text{mit} \quad f(x,y) = \ln(\sqrt{x^2+y^2}),$$

die für alle Maschinenzahlen $\widetilde{x}, \widetilde{y} \in S(2,53)$, mit $|\widetilde{x}| + |\widetilde{y}| > 0$ auszuwerten ist. Ohne Einschränkung der Allgemeinheit kann dabei $0 \le y \le x$ vorausgesetzt werden. $f(x,y)$ ist dabei der Logarithmus des Abstandes eines Punktes $P(x,y)$ der Ebene zum Ursprung bzw. der Realteil der komplexwertigen Logarithmusfunktion.

Bezeichnet man mit $\widetilde{f}(\widetilde{x},\widetilde{y})$ den auf der Maschine berechneten Näherungswert für $f(\widetilde{x},\widetilde{y}) \approx \widetilde{f}(\widetilde{x},\widetilde{y})$, so soll ein Schranke $\Delta(f)$ des absoluten Fehlers berechnet werden, falls die Funktionswerte $f(\widetilde{x},\widetilde{y})$ im Unterlaufbereich $U = (-\texttt{MinReal}, +\texttt{Minreal})$ liegen:

$$|\widetilde{f}(\widetilde{x},\widetilde{y}) - f(\widetilde{x},\widetilde{y})| \le \Delta(f) \quad \forall\, f(\widetilde{x},\widetilde{y}) \in U$$

Für alle Funktionswerte $f(\widetilde{x},\widetilde{y}) \notin U$ aus dem normalisierten Bereich ist eine Schranke $\varepsilon(f)$ des relativen Fehlers zu berechnen:

$$\left| \frac{\widetilde{f}(\widetilde{x},\widetilde{y}) - f(\widetilde{x},\widetilde{y})}{f(\widetilde{x},\widetilde{y})} \right| \le \varepsilon(f) \quad \forall\, f(\widetilde{x},\widetilde{y}) \notin U.$$

Die Schranken $\Delta(f)$ und $\varepsilon(f)$ werden zur Implementierung einer Intervallfunktion benötigt, mit der zu vorgegebenen Intervallargumenten $X = [\widetilde{x}_1, \widetilde{x}_2]$ und $Y = [\widetilde{y}_1, \widetilde{y}_2]$ der entsprechende Wertebereich $W_f = \{r \in I\!R \mid r = f(x,y) \wedge x \in X \wedge y \in Y\}$ von $f(x,y)$ garantiert eingeschlossen wird. Dazu benötigt man eine Näherungsfunktion $\widetilde{f}(\widetilde{x},\widetilde{y}) \approx f(\widetilde{x},\widetilde{y})$, die wegen der Monotonie von $\ln(\sqrt{x^2+y^2})$ bez. $s = x^2+y^2$ mit den entsprechenden Punktargumenten $\widetilde{x}_1, \widetilde{y}_1$ bzw. $\widetilde{x}_2, \widetilde{y}_2$ aufzurufen ist und deren Implementierung zusammen mit einer Fehlerabschätzung im nächsten Unterabschnitt ausführlich behandelt wird.

8.2.1 Punktargumente

Zur Implementierung von $f(x,y)$ wird der Bereich der Quadratsummen $s := x^2 + y^2$ in sieben Teilintervalle A_i unterteilt, und in jedem A_i wird $f(x,y)$ durch eine geeignete Funktion $f_i(x,y) \equiv f(x,y) = \ln(\sqrt{x^2+y^2})$ realisiert:

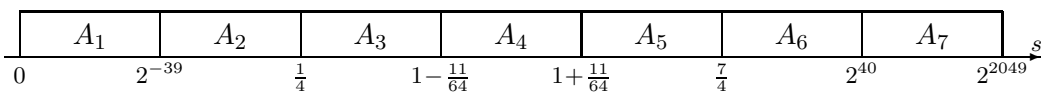


Abbildung 8.1: Teilintervalle A_i der Quadratsummen $s := \widetilde{x}^2 + \widetilde{y}^2$

Die Funktionen $f_i(x,y)$ werden für Punktargumente $\widetilde{x},\widetilde{y} \in S(2,53)$ auf der Maschine ausgewertet und liefern die Näherungen $\widetilde{f_i}(\widetilde{x},\widetilde{y}) \approx f(\widetilde{x},\widetilde{y}) = \ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2})$. In der folgenden Tabelle sind die Funktionen $f_i(x,y)$ zusammengestellt:

A_i	$f_i(x,y) = f(x,y) = \ln(\sqrt{x^2+y^2}), \quad 0 \le y \le x, \quad \lvert y\rvert + \lvert x\rvert > 0;$
A_1	$f_1(x,y) = -N\cdot\ln(2) + \left[\ln(2^N\cdot x) + \frac{1}{2}\ln\left(1+\left(\frac{y}{x}\right)^2\right)\right],\ N \in \mathbb{N}$
A_2	$f_2(x,y) = \frac{1}{2}\ln(x^2+y^2), \quad x^2+y^2$ exakt im Akku berechnen.
A_3	$f_3(x,y) = \frac{1}{2}\cdot\left[\ln(r) + \ln(1+\frac{r_1}{r})\right], \quad x^2+y^2 = r + r_1;$
A_4	$f_4(x,y) = \ln(1+[x^2+y^2-1]), \quad x^2+y^2-1$ exakt im Akku berechnen.
A_5	$f_5(x,y) = \frac{1}{2}\cdot\left[\ln(r) + \ln(1+\frac{r_1}{r})\right], \quad x^2+y^2 = r + r_1;$
A_6	$f_6(x,y) = \frac{1}{2}\ln(x^2+y^2), \quad x^2+y^2$ exakt im Akku berechnen.
A_7	$f_7(x,y) = N\cdot\ln(2) + \left[\ln(2^{-N}\cdot x) + \frac{1}{2}\ln\left(1+\left(\frac{y}{x}\right)^2\right)\right],\ N \in \mathbb{N}$

Tabelle 8.1: $f_i(x,y)$ zur Auswertung von $f(x,y)$ in den Bereichen A_i

Hinweise:

1. Die beiden Bereiche A_1 und A_7 werden in weitere Teilintervalle unterteilt, in denen das $N \in \mathbb{N}$ jeweils so zu wählen ist, dass die Auswertefehler von $f_1(x,y)$ und $f_7(x,y)$ dabei möglichst klein ausfallen. Im Falle $y \ll x$ kann der Summand $\frac{1}{2}\ln(1+\left(\frac{y}{x}\right)^2)$ in $f_1(x,y)$ bzw. $f_7(x,y)$ vernachlässigt werden. Die jeweiligen Unterteilungen von A_1 und A_7 findet man in den Tabellen auf Seite 230 und 254.

2. Bei der Auswertung von $f_2(x,y)$ und $f_6(x,y)$ wird die Quadratsumme $\widetilde{x}^2+\widetilde{y}^2$ rundungsfehlerfrei im Akkumulator berechnet, der danach mit der relativen Fehlerschranke $\varepsilon_0 = 2^{-53}$ ins *double*-Format ausgelesen wird.

3. Bei der Auswertung von $f_3(x,y)$ und $f_5(x,y)$ wird $\widetilde{x}^2+\widetilde{y}^2$ ebenfalls rundungsfehlerfrei im Akkumulator berechnet, der danach zweimal ins *double*-Format ausgelesen wird. Man erhält: $\widetilde{x}^2+\widetilde{y}^2 = r + r_1 \approx r + r1$, mit $r, r1 \in S(2,53)$. Die Maschinenzahl r wird als exakt angesehen, und der absolute Fehler von $r1$ wird abgeschätzt.

4. Die Auswertung von $f_4(x,y)$ erfolgt mit Hilfe der in **C-XSC** implementierten Funktion `lnp1(d)` $\approx \ln(1+$ `d`$)$, wobei $\widetilde{x}^2+\widetilde{y}^2-1$ exakt im Akku ausgewertet wird, der anschließend in die *double*-Variable `d` $\approx \widetilde{x}^2+\widetilde{y}^2-1$ maximalgenau ausgelesen wird. Im Spezialfall $\widetilde{x}=1$ und $\widetilde{y}<2^{-28}$ gilt die Approximation $f_4(\widetilde{x},\widetilde{y})\approx \widetilde{y}^2/2$, wobei die Funktionswerte jetzt sogar in den Unterlaufbereich $U=(-$`MinReal`$,+$`MinReal`$)$ fallen können. In diesem Fall wird nur der absolute Fehler abgeschätzt.

Nach dieser Grobübersicht betrachten wir jetzt die einzelnen Teilbereiche A_i und berechnen für den jeweils angegebenen Algorithmus zur Auswertung von $\ln(\sqrt{x^2+y^2})$ die entsprechende Fehlerschranke des relativen oder absoluten Auswertefehlers. Die Abschätzung der Fehlerschranken erfolgt dabei mit jeweils möglichst kleinen **C-XSC** Programmen, welche die Funktionen der Module `abs_relh, bnd_util, bnd_arh, hilfe` benutzen. Der in **C-XSC** noch ungeübte Leser soll dadurch die Möglichkeit erhalten, die Berechnung der Fehlerschranken möglichst einfach nachvollziehen zu können.

Fehlerabschätzung für $\widetilde{x}^2+\widetilde{y}^2 \in A_1=(0,2^{-39}]$

Bezüglich A_1 erfolgt im Programm `lnx2y2` auf Seite 288 die Abfrage[3]

$$\widetilde{x}<2^{-20} \quad\Longrightarrow\quad \widetilde{x}^2+\widetilde{y}^2<2\cdot 2^{-40}=2^{-39} \tag{8.11}$$

Die Fehlerabschätzung muss daher so durchgeführt werden, dass die Ungleichung rechts in (8.11) erfüllt wird. Für $\widetilde{x}<2^{-20}$ benutzen wir nach Tabelle 8.1:

$$f_1(x,y)=-N\cdot\ln(2)+\left[\ln(2^N\cdot x)+\frac{1}{2}\ln\left(1+\left(\frac{y}{x}\right)^2\right)\right],\ N\in\mathbb{N}, \tag{8.12}$$

wobei im Falle $\widetilde{y}/\widetilde{x}\le 2^{-24}$ der letzte Summand vernachlässigt werden kann:

$$f_1(x,y)\approx -N\cdot\ln(2)+\ln(2^N\cdot x),\ N\in\mathbb{N}; \tag{8.13}$$

Die Idee bei der Anwendung von $f_1(x,y)$ besteht nun darin, den ersten Summanden $-N\cdot\ln(2)$ rechts in (8.12) so zu wählen, dass die Bedingung $f_1(x,y)\approx -N\cdot\ln(2)$ möglichst gut erfüllt ist und dass die wertbestimmende Konstante $-N\cdot\ln(2)$ sich im *double*-Format möglichst genau approximieren lässt. Sind beide Bedingungen erfüllt, so liefert die Auswertung der rechten Seite von (8.12) einen nur kleinen Fehler, da zu einem betragsmäßig großen und fast fehlerfreien Summanden ein zwar fehlerbehafteter aber nur kleiner Summand [...] zu addieren ist. Die Grundidee ist also, die

[3] Aus Laufzeitgründen erfolgt diese Abfrage mit Hilfe der durch `expo()` berechneten Exponenten von $\widetilde{x}$, wobei im Programm die Maschinenzahlen $\widetilde{x},\widetilde{y}$ mit `a,b` und deren Exponenten mit `exa,exb` bezeichnet werden, also z.B. $\widetilde{x}=$ `a` $=$ `m` $\cdot 2^{\texttt{exa}}$, mit $\frac{1}{2}\le|$`m`$|<1$. Beachten Sie zusätzlich: $0\le\widetilde{y}\le\widetilde{x}$.

unvermeidbaren Rundungsfehler möglichst ganz in den betragsmäßig kleineren Summanden [...] zu verlagern! Natürlich lässt sich diese Grundidee nur realisieren, wenn man den ganzen Bereich $2^{-1074} \leq \widetilde{x} < 2^{-20}$ in geeignete Teilintervalle B_j zerlegt und in jedem B_j die Zahl $N \in \mathbb{N}$ so wählt, dass die beiden genannten Bedingungen möglichst gut erfüllt werden.

j	B_j	Rel. Fehlerschranke	N	Rel. Schranke bez. $N \cdot \ln(2)$
-1	$[2^{-1074}, 2^{-1022})$	$2.459639 \cdot 10^{-16}$	1067	$5.81871582840441 \cdot 10^{-19}$
0	$[2^{-1022}, 2^{-950})$	$2.464005 \cdot 10^{-16}$	976	$6.78099202380294 \cdot 10^{-19}$
1	$[2^{-950}, 2^{-900})$	$2.493001 \cdot 10^{-16}$	945	$7.44534477863647 \cdot 10^{-19}$
2	$[2^{-900}, 2^{-850})$	$2.462906 \cdot 10^{-16}$	885	$2.19718262365773 \cdot 10^{-18}$
3	$[2^{-850}, 2^{-800})$	$2.446647 \cdot 10^{-16}$	825	$5.56678584903675 \cdot 10^{-18}$
4	$[2^{-800}, 2^{-750})$	$2.488554 \cdot 10^{-16}$	763	$2.44007669626041 \cdot 10^{-18}$
5	$[2^{-750}, 2^{-700})$	$2.472065 \cdot 10^{-16}$	732	$6.78099202380294 \cdot 10^{-19}$
6	$[2^{-700}, 2^{-650})$	$2.447755 \cdot 10^{-16}$	671	$6.78099202380294 \cdot 10^{-19}$
7	$[2^{-650}, 2^{-600})$	$2.552012 \cdot 10^{-16}$	610	$6.78099202380294 \cdot 10^{-19}$
8	$[2^{-600}, 2^{-550})$	$2.673659 \cdot 10^{-16}$	549	$6.78099202380294 \cdot 10^{-19}$
9	$[2^{-550}, 2^{-500})$	$2.547771 \cdot 10^{-16}$	518	$1.91724602509176 \cdot 10^{-18}$
10	$[2^{-500}, 2^{-450})$	$2.677207 \cdot 10^{-16}$	488	$6.78099202380294 \cdot 10^{-19}$
11	$[2^{-450}, 2^{-400})$	$2.591254 \cdot 10^{-16}$	427	$6.78099202380294 \cdot 10^{-19}$
12	$[2^{-400}, 2^{-350})$	$2.677857 \cdot 10^{-16}$	366	$6.78099202380294 \cdot 10^{-19}$
13	$[2^{-350}, 2^{-300})$	$2.690160 \cdot 10^{-16}$	320	$1.42250834110490 \cdot 10^{-18}$
14	$[2^{-300}, 2^{-250})$	$2.959482 \cdot 10^{-16}$	259	$1.91724602509176 \cdot 10^{-18}$
15	$[2^{-250}, 2^{-200})$	$3.041687 \cdot 10^{-16}$	229	$3.61344598803646 \cdot 10^{-18}$
16	$[2^{-200}, 2^{-150})$	$3.290578 \cdot 10^{-16}$	160	$1.42250834110490 \cdot 10^{-18}$
17	$[2^{-150}, 2^{-100})$	$3.429172 \cdot 10^{-16}$	122	$6.78099202380294 \cdot 10^{-19}$
18	$[2^{-100}, 2^{-50})$	$4.295508 \cdot 10^{-16}$	61	$6.78099202380294 \cdot 10^{-19}$
19	$[2^{-50}, 2^{-30})$	$4.189496 \cdot 10^{-16}$	40	$1.42250834110490 \cdot 10^{-18}$
20	$[2^{-30}, 2^{-20})$	$4.142222 \cdot 10^{-16}$	20	$1.42250834110490 \cdot 10^{-18}$

In der Tabelle auf Seite 230 sind für die einzelnen Teilintervalle B_j angegeben der relative Auswertefehler bei Anwendung von (8.12), der geeignete N-Wert und in der letzten Spalte die relative Fehlerschranke, die beim Speichern von $N \cdot \ln(2)$ im *double*-Format entsteht.

Die folgende Tabelle enthält die relativen Fehlerschranken bei Auswertung von (8.13) in den B_j. In der letzten Spalte findet man die absolute Fehlerschranke, die beim Speichern von $N \cdot \ln(2)$ im *double*-Format entsteht.

j	B_j	Rel. Fehlerschranke	N	Abs. Schranke bez. $N \cdot \ln(2)$
0	$[2^{-1022}, 2^{-950})$	$2.386233 \cdot 10^{-16}$	976	$4.58742009063372 \cdot 10^{-16}$
1	$[2^{-950}, 2^{-900})$	$2.403745 \cdot 10^{-16}$	945	$4.87688015582011 \cdot 10^{-16}$
2	$[2^{-900}, 2^{-850})$	$2.394541 \cdot 10^{-16}$	885	$1.34782928257584 \cdot 10^{-15}$
3	$[2^{-850}, 2^{-800})$	$2.401777 \cdot 10^{-16}$	825	$3.18334658073368 \cdot 10^{-15}$
4	$[2^{-800}, 2^{-750})$	$2.413877 \cdot 10^{-16}$	763	$1.29048653144292 \cdot 10^{-15}$
5	$[2^{-750}, 2^{-700})$	$2.398565 \cdot 10^{-16}$	732	$3.44056506797529 \cdot 10^{-16}$
6	$[2^{-700}, 2^{-650})$	$2.388195 \cdot 10^{-16}$	671	$3.15385131231069 \cdot 10^{-16}$
7	$[2^{-650}, 2^{-600})$	$2.450469 \cdot 10^{-16}$	610	$2.86713755664608 \cdot 10^{-16}$
8	$[2^{-600}, 2^{-550})$	$2.523172 \cdot 10^{-16}$	549	$2.58042380098147 \cdot 10^{-16}$
9	$[2^{-550}, 2^{-500})$	$2.460853 \cdot 10^{-16}$	518	$6.88387644547236 \cdot 10^{-16}$
10	$[2^{-500}, 2^{-450})$	$2.533063 \cdot 10^{-16}$	488	$2.29371004531686 \cdot 10^{-16}$
11	$[2^{-450}, 2^{-400})$	$2.490270 \cdot 10^{-16}$	427	$2.00699628965226 \cdot 10^{-16}$
12	$[2^{-400}, 2^{-350})$	$2.549860 \cdot 10^{-16}$	366	$1.72028253398765 \cdot 10^{-16}$
13	$[2^{-350}, 2^{-300})$	$2.571003 \cdot 10^{-16}$	320	$3.15522446707157 \cdot 10^{-16}$
14	$[2^{-300}, 2^{-250})$	$2.741487 \cdot 10^{-16}$	259	$3.44193822273618 \cdot 10^{-16}$
15	$[2^{-250}, 2^{-200})$	$2.816550 \cdot 10^{-16}$	229	$5.73564826805304 \cdot 10^{-16}$
16	$[2^{-200}, 2^{-150})$	$2.991208 \cdot 10^{-16}$	160	$1.57761223353579 \cdot 10^{-16}$
17	$[2^{-150}, 2^{-100})$	$3.133037 \cdot 10^{-16}$	122	$5.73427511329215 \cdot 10^{-17}$
18	$[2^{-100}, 2^{-50})$	$3.888831 \cdot 10^{-16}$	61	$2.86713755664608 \cdot 10^{-17}$
19	$[2^{-50}, 2^{-30})$	$4.088072 \cdot 10^{-16}$	40	$3.94403058383947 \cdot 10^{-17}$
20	$[2^{-30}, 2^{-20})$	$4.524090 \cdot 10^{-16}$	20	$1.97201529191974 \cdot 10^{-17}$

Berechnung der N-Werte

Am Beispiel des Intervalls $B_{-1} = [2^{-1074}, 2^{-1022})$ wird jetzt gezeigt, wie der günstigste Wert $N = 1067$ zustande kommt. Beachten Sie bitte zunächst, dass in (8.12) der Summand $\ln(2^N \cdot x)$ in B_{-1} nur für $52 \leq N \leq 2045$ ohne Fehlermeldung ausgewertet werden kann, da die **C-XSC** Funktion `real ln(const real&)` für Argumente aus dem denormalisierten Bereich $(0, 2^{-1022})$ nicht definiert ist.

Nach den Bemerkungen von Seite 229 unten suchen wir daher in $52 \leq N \leq 2045$ zunächst solche N-Werte, für die $N \cdot \ln(2)$ im *double*-Format mit einem möglichst kleinen relativen Fehler gespeichert werden kann. Das folgende Programm `Nln2.cpp`

```
// Program Nln2.cpp calculats the optimal N within a given
// interval for N, so that N*ln(2) can be stored as a real number
// with a minimal relative error.
// Output:  N, absolute and relative error, approximation of N*ln(2)
//          and optimal inclusion of N*ln(2);

#include <iostream>     // for cout
#include <l_rmath.hpp>  // for ln(..) with l_interval argument
#include "bnd_util.hpp" // for IEEE2frac_int(..)

using namespace cxsc;
using namespace std;

int main()
{
    int N1,N2,Nopt,N;
    real Nln_2,abs_,minr=MaxReal,mina=MaxReal,rel,z,n,z1,n1,z2,n2;
    stagprec = 4;  // Precision of the staggered arithmetic
    l_interval li_ln2=ln( l_interval(2) ),li_Nln2;
    interval zz;  // For inclusions in the double format
    cout << endl;
    cout << "Calculating an optimal N within an interval [N1,N2] to"
         << endl;
    cout << "store N*ln(2) as one real number with a minimal "
         << "relative error:" << endl;
    cout << "N1 = ? ";  cin >> N1;
    cout << "N2 = ? ";  cin >> N2;
    if (N2<N1) N2 = N1+1;

    N=N1;
```

```
    l_real ln2 = ln( l_real(2) ), Nln2;
    while (N<=N2)
    {
        Nln2  = N * ln2;  // Nln2: Staggered approximation of N*ln2
        Nln_2 = Nln2[1];  // Optimal real-approximation for N*ln(2)
        li_Nln2 = N * li_ln2;  // li_Nln2: Inclusion of N*ln(2)
        zz = li_Nln2 - Nln_2;  // zz: Inclusion of  N*ln(2)-Nln_2
        abs_ = Sup(abs(zz)); // abs_: upper bound of the abs. error
        zz = li_Nln2;        // zz: Inclusion of N*ln(2)
        rel = divu(abs_,Inf(abs(zz))); // rel: relative error bound

        if (rel<minr) {
            minr = rel; mina = abs_; Nopt = N;
            IEEE2frac_int(Nln_2,z,n);
            // zz: Inclusion of Nopt*ln(2)
            IEEE2frac_int(Inf(zz),z1,n1); // Inf(zz) = z1/n1
            IEEE2frac_int(Sup(zz),z2,n2); // Sup(zz) = z2/n2
        }
        N++;
    }
    cout << "With the optimal N = " << Nopt << " we get:" << endl;
    cout << SetPrecision(14,14) << RndUp
         << "Optimal relative error < " << minr << endl;
    cout << "Optimal absolute error < " << mina << endl;
    cout << "Optimal approximation of " << Nopt
         << "*ln(2) with the quotient:"  << endl;
    cout << SetPrecision(1,1) << Fixed << "          "
         << z << " / " << n << endl;
    cout << "Optimal inclusion of " << Nopt
         << "*ln(2) with the interval [a,b]:"  << endl;
    cout << "a = " << z1 << " / " << n1 << endl;
    cout << "b = " << z2 << " / " << n2 << endl << endl;
}
```

liefert zu einem vorgegebenen Intervall $[N_1, N_2]$ den N-Wert, mit dem $N \cdot \ln(2)$ im *double*-Format optimal gespeichert werden kann. Zusätzlich erhält man den entsprechenden absoluten bzw. relativen Fehler und einen Quotienten $z/n \in S(2, 53)$ für die berechnete Approximation von $N \cdot \ln(2) \approx z/n$. Für die Abschätzung des Auswertefehlers nach (8.12) wird zusätzlich noch eine garantierte Einschließung von $N \cdot \ln(2)$ angegeben: $z1/n1 \leq N \cdot \ln(2) \leq z2/n2$. Die obigen Programmberechnungen basieren auf einer staggered Intervall-Arithmetik, die ausführlich kommentiert wird.

In der folgenden Tabelle 8.2 sind einige N-Werte mit ihren absoluten und relativen Fehlerschranken zusammen mit der Approximation für $N \cdot \ln(2) \approx \frac{z}{n} \in S(2,53)$ angegeben. Die Berechnungen wurden mit dem Programm `Nln2.cpp` durchgeführt.

N_i	$N_i \cdot \ln(2) \approx \frac{z}{n}$	Relativer Fehler	Absoluter Fehler
61	$\frac{5950659388407608.0}{140737488355328.0}$	$6.78099202380294 \cdot 10^{-19}$	$2.86713755664608 \cdot 10^{-17}$
183	$\frac{8925989082611412.0}{70368744177664.0}$	$6.78099202380294 \cdot 10^{-19}$	$8.60141266993823 \cdot 10^{-17}$
1067	$\frac{6505485212531678.0}{8796093022208.0}$	$5.81871582840441 \cdot 10^{-19}$	$4.30345264449089 \cdot 10^{-16}$
1189	$\frac{7249317636082629.0}{8796093022208.0}$	$4.52589466947313 \cdot 10^{-19}$	$3.73002513316168 \cdot 10^{-16}$
1220	$\frac{7438324235509510.0}{8796093022208.0}$	$6.78099202380294 \cdot 10^{-19}$	$5.73427511329215 \cdot 10^{-16}$
1311	$\frac{7993150059633580.0}{8796093022208.0}$	$3.47369011067856 \cdot 10^{-19}$	$3.15659762183246 \cdot 10^{-16}$
1433	$\frac{8736982483184531.0}{8796093022208.0}$	$2.60064669099486 \cdot 10^{-19}$	$2.58317011050325 \cdot 10^{-16}$
1799	$\frac{5484239876918692.0}{4398046511104.0}$	$6.91986452186632 \cdot 10^{-20}$	$8.62887576515599 \cdot 10^{-17}$
2043	$\frac{6228072300469643.0}{4398046511104.0}$	$2.00527864084272 \cdot 10^{-20}$	$2.83967446142832 \cdot 10^{-17}$

Tabelle 8.2: Optimale $N_i \in [52, 2045]$ zur Speicherung von $N_i \cdot \ln(2) \approx z/n$.

Mit $N = 2043$ kann damit die Konstante $2043 \cdot \ln(2)$ durch die Maschinenzahl

$$\frac{z}{n} = \frac{6228072300469643.0}{4398046511104.0} \approx 2043 \cdot \ln(2)$$

im *double*-Format mit dem kleinsten relativen Fehler $\varepsilon_0 = 2.00527864084272 \cdot 10^{-20}$ approximiert werden. Dies bedeutet jedoch nicht, dass wir mit $N = 2043$ auch bei Anwendung von (8.12) auf Seite 229 den kleinsten relativen Auswertefehler erhalten. Nach den Bemerkungen von Seite 229 unten muss zusätzlich die Bedingung

$$|-N \cdot \ln(2)| \gg \left| \ln(2^N \cdot x) + \frac{1}{2} \ln\left(1 + \left(\frac{y}{x}\right)^2\right) \right| \tag{8.14}$$

für alle Maschinenzahlen $\widetilde{x} \in B_{-1} = [2^{-1074}, 2^{-1022})$ möglichst gut erfüllt sein. Zur Überprüfung von (8.14) sind in der nachfolgenden Tabelle die Beträge von $-N \ln(2)$ und $\ln(2^N x)$ zusammengestellt. Der Summand $\frac{1}{2}\ln(1 + (\frac{y}{x})^2) \leq \frac{1}{2}\ln(2) = 0.346\ldots$ spielt bei diesen Vergleichen keine Rolle!

N_i	$N_i \cdot \ln(2)$	$\lvert\ln(2^{N_i}x)\rvert \leq$	N_i	$N_i \cdot \ln(2)$	$\lvert\ln(2^{N_i}x)\rvert \leq$
61	42.281...	702.158...	1220	845.63...	137.24...
183	126.85...	617.59...	1311	908.71...	200.31...
1067	739.58...	31.191...	1799	1246.9...	538.57...
1189	824.15...	115.75...	2043	1416.0...	707.70...

Tabelle 8.3: Vergleich von $N_i \cdot \ln(2)$ mit $\lvert\ln(2^{N_i}x)\rvert$ für $x \in [2^{-1074}, 2^{-1022})$

Aus obiger Tabelle 8.3 erkennt man, dass die Bedingung (8.14) für $N = 1067$ am besten erfüllt ist, so dass wir mit $N = 1067$ bei der Maschinenauswertung von (8.12) für $\widetilde{x} \in B_{-1} = [2^{-1074}, 2^{-1022})$ den kleinsten Auswertefehler erwarten können.

Berechnung des Auswertefehlers in B_{-1}

Zur Berechnung einer Oberschranke des relativen Fehlers bei Auswertung von (8.12) auf Seite 229 benutzen wir für $\widetilde{x} \in B_{-1} = [2^{-1074}, 2^{-1022})$ das Programm

```
// Program lnx2y2_Bd.cpp for calculating the relative error by
// evaluating the term
// -N*ln(2) + ln(2^N*x) + 0.5*ln[1+(y/x)^2],
// with N=61,183,1067,1189,1220,1311,1799,2043;
// for minreal=2^(-1074) <= x <= MinReal=2^(-1022)  and  0 <= y <= x.

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real Inf_61 = -5950659388407608.0 / 140737488355328.0;
real Sup_61 = -5950659388407607.0 / 140737488355328.0;
interval ln2_61 = interval(Inf_61,Sup_61);
      // ln2_61 is an optimal inclusion of -61*ln(2);
real rln2_61 = -5950659388407608.0 / 140737488355328.0;
  // rln2_61 is an optimal real approximation for -61*ln(2);
real abs_61 = 2.8671376e-17;
  // abs_61 is an abs. error bound of the above real approximation
```

```
real Inf_183 = -8925989082611412.0 / 70368744177664.0;
real Sup_183 = -8925989082611411.0 / 70368744177664.0;

interval ln2_183 = interval(Inf_183,Sup_183);
      // ln2_183 is an optimal inclusion of -183*ln(2);
real rln2_183 = -8925989082611412.0 / 70368744177664.0;
  // rln2_183 is an optimal real approximation for -183*ln(2);
real abs_183 = 8.60141267e-17;
  // abs_183 is an abs. error bound of the above real approximation

real Inf_1067 = -6505485212531679.0 / 8796093022208.0;
real Sup_1067 = -6505485212531678.0 / 8796093022208.0;
interval ln2_1067 = interval(Inf_1067,Sup_1067);
      // ln2_1067 is an optimal inclusion of -1067*ln(2);
real rln2_1067 = -6505485212531678.0 / 8796093022208.0;
  // rln2_1067 is an optimal real approximation for -1067*ln(2);
real abs_1067 = 4.3034526445e-16;
  // abs_1067 is an abs. error bound of the above real approximation

real Inf_1189 = -7249317636082630.0 / 8796093022208.0;
real Sup_1189 = -7249317636082629.0 / 8796093022208.0;
interval ln2_1189 = interval(Inf_1189,Sup_1189);
      // ln2_1189 is an optimal inclusion of -1189*ln(2);
real rln2_1189 = -7249317636082629.0 / 8796093022208.0;
  // rln2_1189 is an optimal real approximation for -1189*ln(2);
real abs_1189 = 3.73002513317e-16;
  // abs_1189 is an abs. error bound of the above real approximation

real Inf_1220 = -7438324235509510.0 / 8796093022208.0;
real Sup_1220 = -7438324235509509.0 / 8796093022208.0;
interval ln2_1220 = interval(Inf_1220,Sup_1220);
      // ln2_1220 is an optimal inclusion of -1220*ln(2);
real rln2_1220 = -7438324235509510.0 / 8796093022208.0;
  // rln2_1220 is an optimal real approximation for -1220*ln(2);
real abs_1220 = 5.73427511329215E-016;
  // abs_1220 is an abs. error bound of the above real approximation

real Inf_1311 = -7993150059633581.0 / 8796093022208.0;
real Sup_1311 = -7993150059633580.0 / 8796093022208.0;
interval ln2_1311 = interval(Inf_1311,Sup_1311);
```

```
      // ln2_1311 is an optimal inclusion of -1311*ln(2);
real rln2_1311 = -7993150059633580.0 / 8796093022208.0;
  // rln2_1311 is an optimal real approximation for -1311*ln(2);

real abs_1311 = 3.15659762183246E-016;
  // abs_1311 is an abs. error bound of the above real approximation

real Inf_1799 = -5484239876918693.0 / 4398046511104.0;
real Sup_1799 = -5484239876918692.0 / 4398046511104.0;
interval ln2_1799 = interval(Inf_1799,Sup_1799);
      // ln2_1799 is an optimal inclusion of -1799*ln(2);
real rln2_1799 = -5484239876918692.0 / 4398046511104.0;
  // rln2_1799 is an optimal real approximation for -1799*ln(2);
real abs_1799 = 8.62887576515599E-017;
  // abs_1799 is an abs. error bound of the above real approximation

real Inf_2043 = -6228072300469643.0 / 4398046511104.0;
real Sup_2043 = -6228072300469642.0 / 4398046511104.0;
interval ln2_2043 = interval(Inf_2043,Sup_2043);
      // ln2_2043 is an optimal inclusion of -2043*ln(2);
real rln2_2043 = -6228072300469643.0 / 4398046511104.0;
  // rln2_2043 is an optimal real approximation for -2043*ln(2);
real abs_2043 = 2.83967446142832E-017;
  // abs_2043 is an abs. error bound of the above real approximation

real T_x(const interval& Xi, const real& delx)
{
    interval Yi=interval(0,Sup(Xi)),A,B;
    real dela,delb,r,eps;
    abs_divh1(Yi,delx,Xi,delx,A,dela); // y/x
    abs_mulh1(A,dela,A,dela,B,delb);   // (y/x)*(y/x)
    abs_lnp1_abs(B,delb,A,dela);
    times2pown(A,-1);    // Division by 2
    times2pown(dela,-1); // Division by 2
    Yi = Xi;  times2pown(Yi,1067);
    abs_ln_abs(Yi,delx,B,delb);
    abs_addh1(A,dela,B,delb,Yi,r);

    rel_addh1(Yi,r,ln2_1067,abs_1067,A,eps); // eps: rel. error bound
    return eps;
}
```

```
int main()
{
    interval X = interval(comp(0.5,-1073),comp(0.5,-1021));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Für die möglichen Werte $N \in \{61, 183, 1067, 1189, 1220, 1311, 1799, 2043\}$ müssen im obigen Programm in der Funktion `T_x(...)` lediglich die drei entsprechenden Werte eingetragen werden. Im obigen Quelltext ist `T_x(...)` für $N = 1067$ angegeben. In der folgenden Tabelle sind in Abhängigkeit von N die berechneten relativen Fehlerschranken bei Auswertung von (8.12) auf Seite 229 für $\widetilde{x} \in B_{-1} = [2^{-1074}, 2^{-1022})$ zusammengestellt:

N	Rel. Fehler	N	Rel. Fehler	N	Rel. Fehler	N	Rel. Fehler
61	$7.12 \cdot 10^{-16}$	183	$6.53 \cdot 10^{-16}$	1067	$2.46 \cdot 10^{-16}$	1189	$3.08 \cdot 10^{-16}$
1220	$3.24 \cdot 10^{-16}$	1311	$3.70 \cdot 10^{-16}$	1799	$5.83 \cdot 10^{-16}$	2043	$5.17 \cdot 10^{-16}$

Tabelle 8.4: Relative Fehlerschranken in Abhängigkeit von N

Wie erwartet erhalten wir für $N = 1067$ die kleinste Schranke des relativen Fehlers bei Auswertung von (8.12) im Bereich $\widetilde{x} \in B_{-1} = [2^{-1074}, 2^{-1022})$

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_1(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, N = 1067) = 2.459639 \cdot 10^{-16}$$

Hinweise zum Programm `lnx2y2_Bd`

1. Die von N abhängigen Konstanten wurden mit Hilfe des Programms `Nln2.cpp` von Seite 232 berechnet.

2. Die Werte $(\widetilde{y}/\widetilde{x})^2$ liegen für $\widetilde{y} > 0$ und $\widetilde{x} \in B_{-1}$ alle im normalisierten Bereich, da für $\widetilde{y} > 0$ gilt: $\widetilde{y}/\widetilde{x} \geq 2^{-1074}/2^{-1022} = 2^{-52}$.

3. Für $\widetilde{x} \in X = [x_1, x_2]$, mit $x_1 \geq 2^{-1074}$ und $x_2 \leq 2^{-1022}$ gilt: $\widetilde{y} \in Y = [0, x_2]$, d.h. X wird bei der Fehlerabschätzung vorgegeben und muss gegebenenfalls hinreichend fein unterteilt werden.

4. Y/X enthält für alle $\widetilde{x} \in X$ alle Quotienten $\widetilde{y}/\widetilde{x}$, mit $0 \leq \widetilde{y} \leq \widetilde{x}$. Das Intervall Y/X enthält jedoch auch Quotienten $\widetilde{y}/\widetilde{x} > 1$. Diese nicht erwünschte Überschätzung wird aber umso geringer, je kleiner der Durchmesser von X gewählt wird, d.h. eine feine Unterteilung von X ist wirklich notwendig und wird mit dem Parameter `diam=1e-5` gesteuert. Die Funktion `T_x(...)` berechnet für ein solches Teilintervall `Xi` mit dem Rückgabewert `eps` eine Schranke des relativen Auswertefehlers.

5. Die in `T_x(..)` benutzten Funktionen `abs_divh1,abs_mulh1,abs_addh1` und `rel_addh1` werden im Abschnitt 2.5.2 ab Seite 36 ausführlich beschrieben. Informationen zu den Funktionen `abs_lnp1_abs,abs_ln_abs` findet man in den Abschnitten 6.2.3 und 6.2.5 ab Seite 161 bzw. 167.

Berechnung des Auswertefehlers in B_j, $j = 0, 1, \ldots, 20$;

Am Beispiel $\widetilde{x} \in B_{-1}$ haben wir ausführlich beschrieben, wie der günstigste N-Wert und der zugehörige relative Fehler bei Auswertung von (8.12) zu berechnen ist. Mit dem gleichen Verfahren kann man für die Teilintervalle B_j, mit $j = 0, 1, \ldots, 20$ das jeweilige N und den dazugehörigen relativen Fehler bestimmen. Die Ergebnisse sind in den Spalten 3 und 4 der Tabelle von Seite 230 zusammengestellt. Die relativen Fehlerschranken in Spalte 3 wurden mit dem Programm `lnx2y2_Bj` berechnet:

```
// Program lnx2y2_Bj.cpp for calculating the relative error by
// evaluating the term
// -N*ln(2) + [ln(2^N*x) + 0.5*ln[1+(y/x)^2]];
// for 2^-1022 <= x <= 2^-20  and  0 <= y <= x.
// ************************************************************

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real Inf_976 = 5950659388407607.0 / 8796093022208.0;
real Sup_976 = 5950659388407608.0 / 8796093022208.0;
interval ln2_976 = -interval(Inf_976,Sup_976);
      // ln2_976 is an optimal inclusion of -976*ln(2);
real rln2_976 = -5950659388407608.0 / 8796093022208.0;
  // rln2_976 is an optimal real approximation for -976*ln(2);
real abs_976 = 4.58742009063372E-016;
```

```
  // abs_976 is an abs. error bound of the above real approximation

real Inf_945 = 5761652788980727.0 / 8796093022208.0;
real Sup_945 = 5761652788980728.0 / 8796093022208.0;
interval ln2_945 = -interval(Inf_945,Sup_945);
      // ln2_945 is an optimal inclusion of -945*ln(2);
real rln2_945 = -5761652788980727.0 / 8796093022208.0;
  // rln2_945 is an optimal real approximation for -945*ln(2);
real abs_945 = 4.87688015582011E-016;
  // abs_945 is an abs. error bound of the above real approximation

real Inf_885 = 5395833564283537.0 / 8796093022208.0;
real Sup_885 = 5395833564283538.0 / 8796093022208.0;
interval ln2_885 = -interval(Inf_885,Sup_885);
      // ln2_885 is an optimal inclusion of -885*ln(2);
real rln2_885 = -5395833564283538.0 / 8796093022208.0;
  // rln2_885 is an optimal real approximation for -885*ln(2);
real abs_885 = 1.34782928257584E-015;
  // abs_885 is an abs. error bound of the above real approximation

real Inf_825 = 5030014339586348.0 / 8796093022208.0;
real Sup_825 = 5030014339586349.0 / 8796093022208.0;
interval ln2_825 = -interval(Inf_825,Sup_825);
      // ln2_825 is an optimal inclusion of -825*ln(2);
real rln2_825 = -5030014339586349.0 / 8796093022208.0;
  // rln2_825 is an optimal real approximation for -825*ln(2);
real abs_825 = 3.18334658073368E-015;
  // abs_825 is an abs. error bound of the above real approximation

real Inf_763 = 4652001140732586.0 / 8796093022208.0;
real Sup_763 = 4652001140732587.0 / 8796093022208.0;
interval ln2_763 = -interval(Inf_763,Sup_763);
      // ln2_763 is an optimal inclusion of -763*ln(2);
real rln2_763 = -4652001140732587.0 / 8796093022208.0;
  // rln2_763 is an optimal real approximation for -763*ln(2);
real abs_763 = 1.29048653144292E-015;
  // abs_763 is an abs. error bound of the above real approximation

real Inf_732 = 8925989082611411.0 / 17592186044416.0;
real Sup_732 = 8925989082611412.0 / 17592186044416.0;
interval ln2_732 = -interval(Inf_732,Sup_732);
```

```
      // ln2_732 is an optimal inclusion of -732*ln(2);

real rln2_732 = -8925989082611412.0 / 17592186044416.0;
  // rln2_732 is an optimal real approximation for -732*ln(2);
real abs_732 = 3.44056506797529E-016;
  // abs_732 is an abs. error bound of the above real approximation

real Inf_671 = 8182156659060460.0 / 17592186044416.0;
real Sup_671 = 8182156659060461.0 / 17592186044416.0;
interval ln2_671 = -interval(Inf_671,Sup_671);
      // ln2_671 is an optimal inclusion of -671*ln(2);
real rln2_671 = -8182156659060461.0 / 17592186044416.0;
  // rln2_671 is an optimal real approximation for -671*ln(2);
real abs_671 = 3.15385131231069E-016;
  // abs_671 is an abs. error bound of the above real approximation

real Inf_610 = 7438324235509509.0 / 17592186044416.0;
real Sup_610 = 7438324235509510.0 / 17592186044416.0;
interval ln2_610 = -interval(Inf_610,Sup_610);
      // ln2_610 is an optimal inclusion of -610*ln(2);
real rln2_610 = -7438324235509510.0 / 17592186044416.0;
  // rln2_610 is an optimal real approximation for -610*ln(2);
real abs_610 = 2.86713756E-016;
  // abs_610 is an abs. error bound of the above real approximation

real Inf_549 = 6694491811958558.0 / 17592186044416.0;
real Sup_549 = 6694491811958559.0 / 17592186044416.0;
interval ln2_549 = -interval(Inf_549,Sup_549);
      // ln2_549 is an optimal inclusion of -549*ln(2);
real rln2_549 = -6694491811958559.0 / 17592186044416.0;
  // rln2_549 is an optimal real approximation for -549*ln(2);
real abs_549 = 2.58042381E-016;
  // abs_549 is an abs. error bound of the above real approximation

real Inf_518 = 6316478613104797.0 / 17592186044416.0;
real Sup_518 = 6316478613104798.0 / 17592186044416.0;
interval ln2_518 = -interval(Inf_518,Sup_518);
      // ln2_518 is an optimal inclusion of -518*ln(2);
real rln2_518 = -6316478613104797.0 / 17592186044416.0;
  // rln2_518 is an optimal real approximation for -518*ln(2);
real abs_518 = 6.88387645E-016;
```

```
  // abs_518 is an abs. error bound of the above real approximation

real Inf_488 = 5950659388407607.0 / 17592186044416.0;
real Sup_488 = 5950659388407608.0 / 17592186044416.0;
interval ln2_488 = -interval(Inf_488,Sup_488);
      // ln2_488 is an optimal inclusion of 488*ln(2);
real rln2_488 = -5950659388407608.0 / 17592186044416.0;
  // rln2_488 is an optimal real approximation for -488*ln(2);
real abs_488 = 2.29371004531686E-016;
  // abs_488 is an abs. error bound of the above real approximation

real Inf_427 = 5206826964856656.0 / 17592186044416.0;
real Sup_427 = 5206826964856657.0 / 17592186044416.0;
interval ln2_427 = -interval(Inf_427,Sup_427);
      // ln2_427 is an optimal inclusion of 427*ln(2);
real rln2_427 = -5206826964856657.0 / 17592186044416.0;
  // rln2_427 is an optimal real approximation for -427*ln(2);
real abs_427 = 2.00699628965226E-016;
  // abs_427 is an abs. error bound of the above real approximation

real Inf_366 = 8925989082611411.0 / 35184372088832.0;
real Sup_366 = 8925989082611412.0 / 35184372088832.0;
interval ln2_366 = -interval(Inf_366,Sup_366);
      // ln2_366 is an optimal inclusion of 366*ln(2);
real rln2_366 = -8925989082611412.0 / 35184372088832.0;
  // rln2_366 is an optimal real approximation for -366*ln(2);
real abs_366 = 1.72028253398765E-016;
  // abs_366 is an abs. error bound of the above real approximation

real Inf_320 = 7804143460206699.0 / 35184372088832.0;
real Sup_320 = 7804143460206700.0 / 35184372088832.0;
interval ln2_320 = -interval(Inf_320,Sup_320);
      // ln2_320 is an optimal inclusion of 320*ln(2);
real rln2_320 = -7804143460206699.0 / 35184372088832.0;
  // rln2_320 is an optimal real approximation for -320*ln(2);
real abs_320 = 3.15522446707157E-016;
  // abs_320 is an abs. error bound of the above real approximation

real Inf_259 = 6316478613104797.0 / 35184372088832.0;
real Sup_259 = 6316478613104798.0 / 35184372088832.0;
interval ln2_259 = -interval(Inf_259,Sup_259);
```

```
      // ln2_259 is an optimal inclusion of 259*ln(2);

real rln2_259 = -6316478613104797.0 / 35184372088832.0;
  // rln2_259 is an optimal real approximation for -259*ln(2);
real abs_259 = 3.44193822273618E-016;
  // abs_259 is an abs. error bound of the above real approximation

real Inf_229 = 5584840163710418.0 / 35184372088832.0;
real Sup_229 = 5584840163710419.0 / 35184372088832.0;
interval ln2_229 = -interval(Inf_229,Sup_229);
      // ln2_229 is an optimal inclusion of 229*ln(2);
real rln2_229 = -5584840163710419.0 / 35184372088832.0;
  // rln2_229 is an optimal real approximation for -229*ln(2);
real abs_229 = 5.73564826805304E-016;
  // abs_229 is an abs. error bound of the above real approximation

real Inf_160 = 7804143460206699.0 / 70368744177664.0;
real Sup_160 = 7804143460206700.0 / 70368744177664.0;
interval ln2_160 = -interval(Inf_160,Sup_160);
      // ln2_160 is an optimal inclusion of 160*ln(2);
real rln2_160 = -7804143460206699.0 / 70368744177664.0;
  // rln2_160 is an optimal real approximation for -160*ln(2);
real abs_160 = 1.57761223353579E-016;
  // abs_160 is an abs. error bound of the above real approximation

real Inf_122 = 5950659388407607.0 / 70368744177664.0;
real Sup_122 = 5950659388407608.0 / 70368744177664.0;
interval ln2_122 = -interval(Inf_122,Sup_122);
      // ln2_122 is an optimal inclusion of 122*ln(2);
real rln2_122 = -5950659388407608.0 / 70368744177664.0;
  // rln2_122 is an optimal real approximation for -122*ln(2);
real abs_122 = 5.73427511329215E-017;
  // abs_122 is an abs. error bound of the above real approximation

real Inf_61 = 5950659388407607.0 / 140737488355328.0;
real Sup_61 = 5950659388407608.0 / 140737488355328.0;
interval ln2_61 = -interval(Inf_61,Sup_61);
      // ln2_61 is an optimal inclusion of 61*ln(2);
real rln2_61 = -5950659388407608.0 / 140737488355328.0;
  // rln2_61 is an optimal real approximation for -61*ln(2);
real abs_61 = 2.86713755664608E-017;
```

```
  // abs_61 is an abs. error bound of the above real approximation

real Inf_40 = 7804143460206699.0 / 281474976710656.0;
real Sup_40 = 7804143460206700.0 / 281474976710656.0;
interval ln2_40 = -interval(Inf_40,Sup_40);
      // ln2_40 is an optimal inclusion of 40*ln(2);
real rln2_40 = -7804143460206699.0 / 281474976710656.0;
  // rln2_40 is an optimal real approximation for -40*ln(2);
real abs_40 = 3.94403058383947E-017;
  // abs_40 is an abs. error bound of the above real approximation

real Inf_20 = 7804143460206699.0 / 562949953421312.0;
real Sup_20 = 7804143460206700.0 / 562949953421312.0;
interval ln2_20 = -interval(Inf_20,Sup_20);
      // ln2_20 is an optimal inclusion of 20*ln(2);
real rln2_20 = -7804143460206699.0 / 562949953421312.0;
  // rln2_20 is an optimal real approximation for -20*ln(2);
real abs_20 = 1.97201529191974E-017;
  // abs_20 is an abs. error bound of the above real approximation

real t = comp(0.5,-23),tt;  // t = 2^(-24);

real T_x(const interval& Xi, const real& delx)
// Summation order:  -N*ln(2) + [0.5*ln(1+(y/x)^2) + ln(2^N*x)]
{
    tt = muld(Inf(Xi),t);  // Inf(Xi)*2^(-24) rounded downwards
    interval Yi=interval(tt,Sup(Xi)),A,B;
    real dela,delb,r,eps;
    abs_divh1(Yi,delx,Xi,delx,A,dela); // y/x
    abs_mulh1(A,dela,A,dela,B,delb);   // (y/x)*(y/x)
    abs_lnp1_abs(B,delb,A,dela);
    times2pown(A,-1);    // Division by 2
    times2pown(dela,-1); // Division by 2

    Yi = Xi;  times2pown(Yi,61);
    abs_ln_abs(Yi,delx,B,delb); // ln(2^N*x)
    abs_addh1(A,dela,B,delb,Yi,r);  // [ ... ] ready

    rel_addh1(Yi,r,ln2_61,abs_61,A,eps); // eps: rel. error bound
    return eps;
}
```

```
int main()
{
    interval X = interval(comp(0.5,-99),comp(0.5,-49));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Hinweise zum Programm `lnx2y2_Bj`

1. Die von N abhängigen Konstanten wurden mit Hilfe des Programms `Nln2.cpp` von Seite 232 berechnet.

2. Für $\widetilde{x} \in X = [x_1, x_2]$, mit $x_1 \geq 2^{-1022}$ und $x_2 \leq 2^{-20}$ gilt stets die Aussage: $\widetilde{y} \in Y = (x_1 \cdot 2^{-24}, x_2]$, da im Falle $\widetilde{y} \leq x_1 \cdot 2^{-24}$ in (8.12) der dritte Summand vernachlässigt wird und damit (8.13) zur Anwendung kommt. Man kann dann leicht zeigen, dass die Werte $(\widetilde{y}/\widetilde{x})^2$ alle im normalisierten Bereich liegen. Für das Intervall B_0 gilt z.B. $\widetilde{y}/\widetilde{x} \geq 2^{-24} \cdot 2^{-1022}/2^{-950} = 2^{-96}$. Beachten Sie bitte, dass bei der Fehlerabschätzung die Berechnung von $x_1 \cdot 2^{-24}$ durch den Aufruf von `tt = muld(Inf(Xi),t)` mit gerichteter Rundung nach unten erfolgen muss, um in Y alle möglichen $\widetilde{y}$ erfassen zu können!

3. Y/X enthält für alle $\widetilde{x} \in X$ alle Quotienten $\widetilde{y}/\widetilde{x}$, mit $x_1 \cdot 2^{-24} < \widetilde{y} \leq x_2$. Das Intervall Y/X enthält jedoch auch Quotienten $\widetilde{y}/\widetilde{x} > 1$. Diese unerwünschte Überschätzung wird aber umso geringer, je kleiner der Durchmesser von X gewählt wird, d.h. eine feine Unterteilung von X ist wirklich notwendig und wird mit dem Parameter `diam=1e-5` gesteuert. Die Funktion `T_x(...)` berechnet für ein solches Teilintervall `Xi` mit dem Rückgabewert `eps` eine Schranke des relativen Auswertefehlers. Nach dem Quelltext des obigen Programms wird der relative Fehler für das Teilintervall B_{18} mit $N = 61$ berechnet. Für B_{18} erhält man den maximalen Auswertefehler $\varepsilon(f) = 4.295508 \cdot 10^{-16}$.

4. Die in `T_x(..)` benutzten Funktionen `abs_divh1,abs_mulh1,abs_addh1` und `rel_addh1` werden im Abschnitt 2.5.2 ab Seite 36 ausführlich beschrieben. Informationen zu den Funktionen `abs_lnp1_abs,abs_ln_abs` findet man in den Abschnitten 6.2.3 und 6.2.5 ab Seite 161 bzw. 167.

Nach der Tabelle auf Seite 230 liefert das Programm `lnx2y2_Bj` für den relativen Fehler bei Auswertung von (8.12) im Bereich $\widetilde{x} \in [2^{-1022}, 2^{-20})$ die Schranke

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2}) - \widetilde{f}_1(\widetilde{x},\widetilde{y})}{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2})} \right| \leq \varepsilon(f, N=61) = 4.295508 \cdot 10^{-16}$$

Für $\widetilde{x} \in [2^{-1022}, 2^{-20})$ müssen wir jetzt noch den relativen Fehler abschätzen, wenn bei gegebenem $X = [x_1, x_2]$ im Falle $\widetilde{y} \leq x_1 \cdot 2^{-24}$ in (8.12) auf Seite 229 der dritte Summand vernachlässigt wird und damit (8.13) zur Anwendung kommt.

Bevor wir also den relativen Fehler $\varepsilon(g)$ bei Auswertung der Näherungsfunktion $g(x) := -N \cdot \ln(2) + \ln(2^N \cdot x)$ berechnen, wollen wir zunächst den relativen Approximationsfehler abschätzen. Für das Teilintervall $X = [x_1, x_2]$ gilt wegen $y \leq x_1 \cdot 2^{-24}$

$$\begin{aligned}\left|\frac{\frac{1}{2}\ln(1+(\frac{y}{x})^2)}{\frac{1}{2}\ln(x^2+y^2)}\right| &= \frac{\ln(1+(\frac{y}{x})^2)}{|\ln(x^2+y^2)|} \leq \frac{2^{-48}}{|\ln(x^2+y^2)|} \leq \frac{2^{-48}}{|\ln(2x_2^2)|} \\ &= \frac{2^{-48}}{|\ln(2)+2\ln(x_2)|} = \frac{2^{-48}}{-\ln(2)-2\ln(x_2)} = \varepsilon(app)\end{aligned}$$

Nach (7.1) von Seite 186 gilt dann für den relativen Gesamtfehler

$$\varepsilon(f) = \varepsilon(app) + \varepsilon(g) \cdot [1 + \varepsilon(app)]$$

Die Berechnung der Fehlerschranke $\varepsilon(f)$ in den einzelnen Teilintervallen B_j erfolgt mit dem folgenden Programm `lnx2y2_Bj2.cpp`

```
// Program lnx2y2_Bj2.cpp for calculating the relative error eps(g)
// by evaluating the function   g(x) := -N*ln(2) + ln(2^N*x)
// for 2^-1022<=x<= 2^-20.
// Also an upper bound eps(app) of the approximation error is
// calculated and both error bounds are combined to
// eps(f) = eps(app) + eps(g)[1+eps(app)].
// ****************************************************************

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <imath.hpp>    // For ln(interval(2))
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real Inf_976 = 5950659388407607.0 / 8796093022208.0;
real Sup_976 = 5950659388407608.0 / 8796093022208.0;
interval ln2_976 = -interval(Inf_976,Sup_976);
      // ln2_976 is an optimal inclusion of -976*ln(2);
real rln2_976 = -5950659388407608.0 / 8796093022208.0;
  // rln2_976 is an optimal real approximation for -976*ln(2);
```

```
real abs_976 = 4.58742009063372E-016;
  // abs_976 is an abs. error bound of the above real approximation

real Inf_945 = 5761652788980727.0 / 8796093022208.0;
real Sup_945 = 5761652788980728.0 / 8796093022208.0;
interval ln2_945 = -interval(Inf_945,Sup_945);
      // ln2_945 is an optimal inclusion of -945*ln(2);
real rln2_945 = -5761652788980727.0 / 8796093022208.0;
  // rln2_945 is an optimal real approximation for -945*ln(2);
real abs_945 = 4.87688015582011E-016;
  // abs_945 is an abs. error bound of the above real approximation

real Inf_885 = 5395833564283537.0 / 8796093022208.0;
real Sup_885 = 5395833564283538.0 / 8796093022208.0;
interval ln2_885 = -interval(Inf_885,Sup_885);
      // ln2_885 is an optimal inclusion of -885*ln(2);
real rln2_885 = -5395833564283538.0 / 8796093022208.0;
  // rln2_885 is an optimal real approximation for -885*ln(2);
real abs_885 = 1.34782928257584E-015;
  // abs_885 is an abs. error bound of the above real approximation

real Inf_825 = 5030014339586348.0 / 8796093022208.0;
real Sup_825 = 5030014339586349.0 / 8796093022208.0;
interval ln2_825 = -interval(Inf_825,Sup_825);
      // ln2_825 is an optimal inclusion of -825*ln(2);
real rln2_825 = -5030014339586349.0 / 8796093022208.0;
  // rln2_825 is an optimal real approximation for -825*ln(2);
real abs_825 = 3.18334658073368E-015;
  // abs_825 is an abs. error bound of the above real approximation

real Inf_763 = 4652001140732586.0 / 8796093022208.0;
real Sup_763 = 4652001140732587.0 / 8796093022208.0;
interval ln2_763 = -interval(Inf_763,Sup_763);
      // ln2_763 is an optimal inclusion of -763*ln(2);
real rln2_763 = -4652001140732587.0 / 8796093022208.0;
  // rln2_763 is an optimal real approximation for -763*ln(2);
real abs_763 = 1.29048653144292E-015;
  // abs_763 is an abs. error bound of the above real approximation

real Inf_732 = 8925989082611411.0 / 17592186044416.0;
real Sup_732 = 8925989082611412.0 / 17592186044416.0;
```

```
interval ln2_732 = -interval(Inf_732,Sup_732);
      // ln2_732 is an optimal inclusion of -732*ln(2);

real rln2_732 = -8925989082611412.0 / 17592186044416.0;
  // rln2_732 is an optimal real approximation for -732*ln(2);
real abs_732 = 3.44056506797529E-016;
  // abs_732 is an abs. error bound of the above real approximation

real Inf_671 = 8182156659060460.0 / 17592186044416.0;
real Sup_671 = 8182156659060461.0 / 17592186044416.0;
interval ln2_671 = -interval(Inf_671,Sup_671);
      // ln2_671 is an optimal inclusion of -671*ln(2);
real rln2_671 = -8182156659060461.0 / 17592186044416.0;
  // rln2_671 is an optimal real approximation for -671*ln(2);
real abs_671 = 3.15385131231069E-016;
  // abs_671 is an abs. error bound of the above real approximation

real Inf_610 = 7438324235509509.0 / 17592186044416.0;
real Sup_610 = 7438324235509510.0 / 17592186044416.0;
interval ln2_610 = -interval(Inf_610,Sup_610);
      // ln2_610 is an optimal inclusion of -610*ln(2);
real rln2_610 = -7438324235509510.0 / 17592186044416.0;
  // rln2_610 is an optimal real approximation for -610*ln(2);
real abs_610 = 2.86713756E-016;
  // abs_610 is an abs. error bound of the above real approximation

real Inf_549 = 6694491811958558.0 / 17592186044416.0;
real Sup_549 = 6694491811958559.0 / 17592186044416.0;
interval ln2_549 = -interval(Inf_549,Sup_549);
      // ln2_549 is an optimal inclusion of -549*ln(2);
real rln2_549 = -6694491811958559.0 / 17592186044416.0;
  // rln2_549 is an optimal real approximation for -549*ln(2);
real abs_549 = 2.58042381E-016;
  // abs_549 is an abs. error bound of the above real approximation

real Inf_518 = 6316478613104797.0 / 17592186044416.0;
real Sup_518 = 6316478613104798.0 / 17592186044416.0;
interval ln2_518 = -interval(Inf_518,Sup_518);
      // ln2_518 is an optimal inclusion of -518*ln(2);
real rln2_518 = -6316478613104797.0 / 17592186044416.0;
  // rln2_518 is an optimal real approximation for -518*ln(2);
```

```
real abs_518 = 6.88387645E-016;
  // abs_518 is an abs. error bound of the above real approximation

real Inf_488 = 5950659388407607.0 / 17592186044416.0;
real Sup_488 = 5950659388407608.0 / 17592186044416.0;
interval ln2_488 = -interval(Inf_488,Sup_488);
      // ln2_488 is an optimal inclusion of 488*ln(2);
real rln2_488 = -5950659388407608.0 / 17592186044416.0;
  // rln2_488 is an optimal real approximation for -488*ln(2);
real abs_488 = 2.29371004531686E-016;
  // abs_488 is an abs. error bound of the above real approximation

real Inf_427 = 5206826964856656.0 / 17592186044416.0;
real Sup_427 = 5206826964856657.0 / 17592186044416.0;
interval ln2_427 = -interval(Inf_427,Sup_427);
      // ln2_427 is an optimal inclusion of 427*ln(2);
real rln2_427 = -5206826964856657.0 / 17592186044416.0;
  // rln2_427 is an optimal real approximation for -427*ln(2);
real abs_427 = 2.00699628965226E-016;
  // abs_427 is an abs. error bound of the above real approximation

real Inf_366 = 8925989082611411.0 / 35184372088832.0;
real Sup_366 = 8925989082611412.0 / 35184372088832.0;
interval ln2_366 = -interval(Inf_366,Sup_366);
      // ln2_366 is an optimal inclusion of 366*ln(2);
real rln2_366 = -8925989082611412.0 / 35184372088832.0;
  // rln2_366 is an optimal real approximation for -366*ln(2);
real abs_366 = 1.72028253398765E-016;
  // abs_366 is an abs. error bound of the above real approximation

real Inf_320 = 7804143460206699.0 / 35184372088832.0;
real Sup_320 = 7804143460206700.0 / 35184372088832.0;
interval ln2_320 = -interval(Inf_320,Sup_320);
      // ln2_320 is an optimal inclusion of 320*ln(2);
real rln2_320 = -7804143460206699.0 / 35184372088832.0;
  // rln2_320 is an optimal real approximation for -320*ln(2);
real abs_320 = 3.15522446707157E-016;
  // abs_320 is an abs. error bound of the above real approximation

real Inf_259 = 6316478613104797.0 / 35184372088832.0;
real Sup_259 = 6316478613104798.0 / 35184372088832.0;
```

```
interval ln2_259 = -interval(Inf_259,Sup_259);
      // ln2_259 is an optimal inclusion of 259*ln(2);

real rln2_259 = -6316478613104797.0 / 35184372088832.0;
  // rln2_259 is an optimal real approximation for -259*ln(2);
real abs_259 = 3.44193822273618E-016;
  // abs_259 is an abs. error bound of the above real approximation

real Inf_229 = 5584840163710418.0 / 35184372088832.0;
real Sup_229 = 5584840163710419.0 / 35184372088832.0;
interval ln2_229 = -interval(Inf_229,Sup_229);
      // ln2_229 is an optimal inclusion of 229*ln(2);
real rln2_229 = -5584840163710419.0 / 35184372088832.0;
  // rln2_229 is an optimal real approximation for -229*ln(2);
real abs_229 = 5.73564826805304E-016;
  // abs_229 is an abs. error bound of the above real approximation

real Inf_160 = 7804143460206699.0 / 70368744177664.0;
real Sup_160 = 7804143460206700.0 / 70368744177664.0;
interval ln2_160 = -interval(Inf_160,Sup_160);
      // ln2_160 is an optimal inclusion of 160*ln(2);
real rln2_160 = -7804143460206699.0 / 70368744177664.0;
  // rln2_160 is an optimal real approximation for -160*ln(2);
real abs_160 = 1.57761223353579E-016;
  // abs_160 is an abs. error bound of the above real approximation

real Inf_122 = 5950659388407607.0 / 70368744177664.0;
real Sup_122 = 5950659388407608.0 / 70368744177664.0;
interval ln2_122 = -interval(Inf_122,Sup_122);
      // ln2_122 is an optimal inclusion of 122*ln(2);
real rln2_122 = -5950659388407608.0 / 70368744177664.0;
  // rln2_122 is an optimal real approximation for -122*ln(2);
real abs_122 = 5.73427511329215E-017;
  // abs_122 is an abs. error bound of the above real approximation

real Inf_61 = 5950659388407607.0 / 140737488355328.0;
real Sup_61 = 5950659388407608.0 / 140737488355328.0;
interval ln2_61 = -interval(Inf_61,Sup_61);
      // ln2_61 is an optimal inclusion of 61*ln(2);
real rln2_61 = -5950659388407608.0 / 140737488355328.0;
  // rln2_61 is an optimal real approximation for -61*ln(2);
```

```
real abs_61 = 2.86713755664608E-017;
  // abs_61 is an abs. error bound of the above real approximation

real Inf_40 = 7804143460206699.0 / 281474976710656.0;
real Sup_40 = 7804143460206700.0 / 281474976710656.0;
interval ln2_40 = -interval(Inf_40,Sup_40);
      // ln2_40 is an optimal inclusion of 40*ln(2);
real rln2_40 = -7804143460206699.0 / 281474976710656.0;
  // rln2_40 is an optimal real approximation for -40*ln(2);
real abs_40 = 3.94403058383947E-017;
  // abs_40 is an abs. error bound of the above real approximation

real Inf_20 = 7804143460206699.0 / 562949953421312.0;
real Sup_20 = 7804143460206700.0 / 562949953421312.0;
interval ln2_20 = -interval(Inf_20,Sup_20);
      // ln2_20 is an optimal inclusion of 20*ln(2);
real rln2_20 = -7804143460206699.0 / 562949953421312.0;
  // rln2_20 is an optimal real approximation for -20*ln(2);
real abs_20 = 1.97201529191974E-017;
  // abs_20 is an abs. error bound of the above real approximation

const real app1 = comp(0.5,-47); // app1=2^(-48)

real T_x(const interval& Xi, const real& delx)
// Calculation of the rel. error by summation: -N*ln(2)+ln(2^N*x)
{
    interval A,B;
    real dela,eps;
    B = Xi;  times2pown(B,20);   // B = 2^N*x
    abs_ln_abs(B,delx,A,dela);   // ln(2^N*x)
    rel_addh1(A,dela,ln2_20,abs_20,B,eps); // eps: rel. error bound
    return eps;
}

int main()
{
    interval X = interval(comp(0.5,-29),comp(0.5,-19));
    real r,Sa,bnd,diam = 1e-5,delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd); // bnd = eps(g)
    // Now calculating the approximation error eps(app):
    interval app2,app;
```

```
    app2 = -2*ln( interval(Sup(X)) )-ln(interval(2)); // denominator
    app = app1/app2;// app: Inclusion of the rel. approximation error
    cout << "Rel. approx. error: " << app << endl;  // app = eps(app)
    Sa = Sup(app);     // eps(f) = eps(app) + eps(g)[1+eps(app)]
    r = addu(1,Sa);
    r = mulu(r,bnd);
    bnd = addu(r,Sa);  // bnd = eps(f)
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Hinweise zum Programm `lnx2y2_Bj`

1. Die von N abhängigen Konstanten wurden mit Hilfe des Programms `Nln2.cpp` von Seite 232 berechnet.

2. Die Funktion `T_x(...)` berechnet für ein Teilintervall `Xi` mit dem Rückgabewert `eps` eine Schranke des relativen Auswertefehlers $\varepsilon(g)$. Nach dem Quelltext des obigen Programms wird der relative Fehler $\varepsilon(f)$ für das Teilintervall B_{20} mit $N = 20$ berechnet. Für B_{20} erhält man nach der Tabelle von Seite 231 die maximale Fehlerschranke $\varepsilon(f) = 4.524090 \cdot 10^{-16}$.

3. Die in `T_x(..)` benutzte Funktion `rel_addh1` wird im Abschnitt 2.5.2 ab Seite 36 ausführlich beschrieben. Informationen zur Funktion `abs_ln_abs` findet man im Abschnitt 6.2.3 ab Seite 161.

Nach der Tabelle auf Seite 231 liefert das Programm `lnx2y2_Bj2` für den relativen Fehler bei Auswertung von (8.13) im Bereich $\widetilde{x} \in [2^{-1022}, 2^{-20})$ die Schranke

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_1(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, N = 20) = 4.524090 \cdot 10^{-16}$$

Zusammenfassung:

Für alle $\widetilde{x} \in [2^{-1074}, 2^{-20})$ und damit für alle $s := \widetilde{x}^2 + \widetilde{y}^2 \in A_1 = (0, 2^{-39})$ gilt bei Auswertung von (8.12) bzw. (8.13) auf Seite 229 für die Schranke $\varepsilon(f, A_1)$ des relativen Fehlers:

(8.15)
$$\boxed{\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_1(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, A_1) = 4.524090 \cdot 10^{-16}}$$

Fehlerabschätzung für $\widetilde{x}^2+\widetilde{y}^2 \in A_7 = [2^{40}, 2^{2049})$

Bezüglich A_7 erfolgt im Programm `lnx2y2` auf Seite 288 die Abfrage[4]

$$\widetilde{x} \geq 2^{20} \quad \Longrightarrow \quad 2^{40} \leq \widetilde{x}^2+\widetilde{y}^2 < 2^{2049} \tag{8.16}$$

Die Fehlerabschätzung muss daher so durchgeführt werden, dass die Ungleichung rechts in (8.16) erfüllt wird[5]. Für $\widetilde{x} \geq 2^{+20}$ benutzen wir nach Tabelle 8.1:

$$f_7(x,y) = N\cdot\ln(2) + \left[\ln(2^{-N}\cdot x) + \frac{1}{2}\ln\left(1+\left(\frac{y}{x}\right)^2\right)\right],\ N\in\mathbb{N}, \tag{8.17}$$

wobei im Falle $\widetilde{y}/\widetilde{x} \leq 2^{-24}$ der letzte Summand vernachlässigt werden kann:

$$f_7(x,y) \approx g(x) := N\cdot\ln(2) + \ln(2^{-N}\cdot x),\ N\in\mathbb{N}; \tag{8.18}$$

Die Idee bei der Anwendung von $f_7(x,y)$ besteht jetzt wieder darin, den Summanden $N\cdot\ln(2)$ rechts in (8.17) so zu wählen, dass die Bedingung $f_7(x,y)\approx N\cdot\ln(2)$ möglichst gut erfüllt ist und dass die wertbestimmende Konstante $N\cdot\ln(2)$ sich im *double*-Format möglichst genau approximieren lässt. Sind beide Bedingungen erfüllt, so liefert die Auswertung der rechten Seite von (8.17) einen nur kleinen Fehler, da zu einem betragsmäßig großen und fast fehlerfreien Summanden ein zwar fehlerbehafteter aber nur kleiner Summand [...] zu addieren ist. Die Grundidee ist also, die unvermeidbaren Rundungsfehler möglichst ganz in den betragsmäßig kleineren Summanden [...] zu verlagern! Natürlich lässt sich diese Grundidee nur realisieren, wenn man den ganzen Bereich $2^{20} \leq \widetilde{x} < 2^{1024}$ in geeignete Teilintervalle C_j zerlegt und in jedem C_j die Zahl $N\in\mathbb{N}$ so wählt, dass die beiden genannten Bedingungen möglichst gut erfüllt werden. Die Berechnung der N-Werte in den einzelnen C_j erfolgt wie auf Seite 232 beschrieben. Die Ergebnisse sind in der Tabelle auf Seite 254 zusammengestellt. Es soll noch betont werden, dass bei der Auswertung von $f_7(x,y)$ an Stelle von $\ln(\sqrt{x^2+y^2})$ für $0 \leq N < 2^{1024}/\ln(2)$ und $x \geq 2^{20}$ kein Overflow enstehen kann!

Der bei Anwendung von (8.18) zu berücksichtigende Approximationsfehler wird für $x\in X=[x_1,x_2]$ wie folgt abgeschätzt:

$$\begin{aligned}\left|\frac{\frac{1}{2}\ln(1+(\frac{y}{x})^2)}{\frac{1}{2}\ln(x^2+y^2)}\right| &= \frac{\ln(1+(\frac{y}{x})^2)}{|\ln(x^2+y^2)|} \leq \frac{2^{-48}}{\ln(x^2+y^2)} \leq \frac{2^{-48}}{\ln(x_1^2+0)} \\ &= \frac{2^{-49}}{\ln(x_1)} = \varepsilon(app)\end{aligned}$$

[4] Aus Laufzeitgründen erfolgt diese Abfrage mit Hilfe der durch `expo()` berechneten Exponenten von $\widetilde{x}$, wobei im Programm die Maschinenzahlen $\widetilde{x},\widetilde{y}$ mit `a,b` und deren Exponenten mit `exa,exb` bezeichnet werden, also z.B. $\widetilde{x} = \mathtt{a} = \mathtt{m}\cdot 2^{\mathtt{exa}}$, mit $\frac{1}{2} \leq |\mathtt{m}| < 1$. Beachten Sie zusätzlich: $0 \leq \widetilde{y} \leq \widetilde{x}$.

[5] Beachten Sie dabei: $\widetilde{x} \leq \mathtt{MaxReal} < 2^{+1024} \ \rightsquigarrow\ \widetilde{x}^2+\widetilde{y}^2 < 2\cdot 2^{2\cdot 1024} = 2^{2049}$.

j	C_j	Rel. Fehlerschranke	N	Abs. Schranke bez. $N \cdot \ln(2)$
0	$[2^{950}, 2^{1024})$	$2.474572 \cdot 10^{-16}$	976	$4.58742009063372 \cdot 10^{-16}$
1	$[2^{900}, 2^{950})$	$2.491616 \cdot 10^{-16}$	945	$4.87688015582011 \cdot 10^{-16}$
2	$[2^{850}, 2^{900})$	$2.461457 \cdot 10^{-16}$	885	$1.34782928257584 \cdot 10^{-15}$
3	$[2^{800}, 2^{850})$	$2.445118 \cdot 10^{-16}$	825	$3.18334658073368 \cdot 10^{-15}$
4	$[2^{750}, 2^{800})$	$2.489775 \cdot 10^{-16}$	763	$1.29048653144292 \cdot 10^{-15}$
5	$[2^{700}, 2^{750})$	$2.478382 \cdot 10^{-16}$	732	$3.44056506797529 \cdot 10^{-16}$
6	$[2^{650}, 2^{700})$	$2.449179 \cdot 10^{-16}$	671	$3.15385131231069 \cdot 10^{-16}$
7	$[2^{600}, 2^{650})$	$2.553465 \cdot 10^{-16}$	610	$2.86713755664608 \cdot 10^{-16}$
8	$[2^{550}, 2^{600})$	$2.675131 \cdot 10^{-16}$	549	$2.58042380098147 \cdot 10^{-16}$
9	$[2^{500}, 2^{550})$	$2.549491 \cdot 10^{-16}$	518	$6.88387644547236 \cdot 10^{-16}$
10	$[2^{450}, 2^{500})$	$2.674233 \cdot 10^{-16}$	488	$2.29371004531686 \cdot 10^{-16}$
11	$[2^{400}, 2^{450})$	$2.588015 \cdot 10^{-16}$	427	$2.00699628965226 \cdot 10^{-16}$
12	$[2^{350}, 2^{400})$	$2.680061 \cdot 10^{-16}$	366	$1.72028253398765 \cdot 10^{-16}$
13	$[2^{300}, 2^{350})$	$2.692661 \cdot 10^{-16}$	320	$3.15522446707157 \cdot 10^{-16}$
14	$[2^{250}, 2^{300})$	$2.961951 \cdot 10^{-16}$	259	$3.44193822273618 \cdot 10^{-16}$
15	$[2^{200}, 2^{250})$	$3.034083 \cdot 10^{-16}$	229	$5.73564826805304 \cdot 10^{-16}$
16	$[2^{150}, 2^{200})$	$3.293453 \cdot 10^{-16}$	160	$1.57761223353579 \cdot 10^{-16}$
17	$[2^{100}, 2^{150})$	$3.412027 \cdot 10^{-16}$	122	$5.73427511329215 \cdot 10^{-17}$
18	$[2^{50}, 2^{100})$	$4.296234 \cdot 10^{-16}$	61	$2.86713755664608 \cdot 10^{-17}$
19	$[2^{30}, 2^{50})$	$4.119673 \cdot 10^{-16}$	40	$3.94403058383947 \cdot 10^{-17}$
20	$[2^{20}, 2^{30})$	$4.147195 \cdot 10^{-16}$	20	$1.97201529191974 \cdot 10^{-17}$

In obiger Tabelle sind die mit dem nachfolgenden Programm `lnx2y2_Cj.cpp` berechneten relativen Fehlerschranken bei Auswertung von (8.17) zusammengestellt. Für C_{18} erhält man mit $N = 61$ die maximale Fehlerschranke:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_7(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, N = 61) = 4.296234 \cdot 10^{-16}$$

```
// Program lnx2y2_Cj.cpp for calculating the relative error by
// evaluating the term
// N*ln(2) + [ln(2^(-N)*x) + 0.5*ln[1+(y/x)^2]];
// for 2^20 <= x <= MaxReal and 0 <= y <= x.

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real Inf_976 = 5950659388407607.0 / 8796093022208.0;
real Sup_976 = 5950659388407608.0 / 8796093022208.0;
interval ln2_976 = interval(Inf_976,Sup_976);
      // ln2_976 is an optimal inclusion of 976*ln(2);
real rln2_976 = 5950659388407608.0 / 8796093022208.0;
  // rln2_976 is an optimal real approximation for 976*ln(2);
real abs_976 = 4.58742009063372E-016;
  // abs_976 is an abs. error bound of the above real approximation

real Inf_945 = 5761652788980727.0 / 8796093022208.0;
real Sup_945 = 5761652788980728.0 / 8796093022208.0;
interval ln2_945 = interval(Inf_945,Sup_945);
      // ln2_945 is an optimal inclusion of 945*ln(2);
real rln2_945 = 5761652788980727.0 / 8796093022208.0;
  // rln2_945 is an optimal real approximation for 945*ln(2);
real abs_945 = 4.87688015582011E-016;
  // abs_945 is an abs. error bound of the above real approximation

real Inf_885 = 5395833564283537.0 / 8796093022208.0;
real Sup_885 = 5395833564283538.0 / 8796093022208.0;
interval ln2_885 = interval(Inf_885,Sup_885);
      // ln2_885 is an optimal inclusion of 885*ln(2);
real rln2_885 = 5395833564283538.0 / 8796093022208.0;
  // rln2_885 is an optimal real approximation for 885*ln(2);
real abs_885 = 1.34782928257584E-015;
  // abs_885 is an abs. error bound of the above real approximation

real Inf_825 = 5030014339586348.0 / 8796093022208.0;
```

```
real Sup_825 = 5030014339586349.0 / 8796093022208.0;
interval ln2_825 = interval(Inf_825,Sup_825);
      // ln2_825 is an optimal inclusion of 825*ln(2);
real rln2_825 = 5030014339586349.0 / 8796093022208.0;
  // rln2_825 is an optimal real approximation for 825*ln(2);
real abs_825 = 3.18334658073368E-015;
  // abs_825 is an abs. error bound of the above real approximation

real Inf_763 = 4652001140732586.0 / 8796093022208.0;
real Sup_763 = 4652001140732587.0 / 8796093022208.0;
interval ln2_763 = interval(Inf_763,Sup_763);
      // ln2_763 is an optimal inclusion of 763*ln(2);
real rln2_763 = 4652001140732587.0 / 8796093022208.0;
  // rln2_763 is an optimal real approximation for 763*ln(2);
real abs_763 = 1.29048653144292E-015;
  // abs_763 is an abs. error bound of the above real approximation

real Inf_732 = 4652001140732586.0 / 8796093022208.0;
real Sup_732 = 4652001140732587.0 / 8796093022208.0;
interval ln2_732 = interval(Inf_732,Sup_732);
      // ln2_732 is an optimal inclusion of 732*ln(2);
real rln2_732 = 4652001140732587.0 / 8796093022208.0;
  // rln2_732 is an optimal real approximation for 732*ln(2);
real abs_732 = 1.29048653144292E-015;
  // abs_732 is an abs. error bound of the above real approximation

real Inf_671 = 8182156659060460.0 / 17592186044416.0;
real Sup_671 = 8182156659060461.0 / 17592186044416.0;
interval ln2_671 = interval(Inf_671,Sup_671);
      // ln2_671 is an optimal inclusion of 671*ln(2);
real rln2_671 = 8182156659060461.0 / 17592186044416.0;
  // rln2_671 is an optimal real approximation for 671*ln(2);
real abs_671 = 3.15385131231069E-016;
  // abs_671 is an abs. error bound of the above real approximation

real Inf_610 = 7438324235509509.0 / 17592186044416.0;
real Sup_610 = 7438324235509510.0 / 17592186044416.0;
interval ln2_610 = interval(Inf_610,Sup_610);
      // ln2_610 is an optimal inclusion of 610*ln(2);
real rln2_610 = 7438324235509510.0 / 17592186044416.0;
```

```
  // rln2_610 is an optimal real approximation for 610*ln(2);
real abs_610 = 2.86713756E-016;
  // abs_610 is an abs. error bound of the above real approximation

real Inf_549 = 6694491811958558.0 / 17592186044416.0;
real Sup_549 = 6694491811958559.0 / 17592186044416.0;
interval ln2_549 = interval(Inf_549,Sup_549);
      // ln2_549 is an optimal inclusion of 549*ln(2);
real rln2_549 = 6694491811958559.0 / 17592186044416.0;
  // rln2_549 is an optimal real approximation for 549*ln(2);
real abs_549 = 2.58042381E-016;
  // abs_549 is an abs. error bound of the above real approximation

real Inf_518 = 6316478613104797.0 / 17592186044416.0;
real Sup_518 = 6316478613104798.0 / 17592186044416.0;
interval ln2_518 = interval(Inf_518,Sup_518);
      // ln2_518 is an optimal inclusion of 518*ln(2);
real rln2_518 = 6316478613104797.0 / 17592186044416.0;
  // rln2_518 is an optimal real approximation for 518*ln(2);
real abs_518 = 6.88387645E-016;
  // abs_518 is an abs. error bound of the above real approximation

real Inf_488 = 5950659388407607.0 / 17592186044416.0;
real Sup_488 = 5950659388407608.0 / 17592186044416.0;
interval ln2_488 = interval(Inf_488,Sup_488);
      // ln2_488 is an optimal inclusion of 488*ln(2);
real rln2_488 = 5950659388407608.0 / 17592186044416.0;
  // rln2_488 is an optimal real approximation for 488*ln(2);
real abs_488 = 2.29371004531686E-016;
  // abs_488 is an abs. error bound of the above real approximation

real Inf_427 = 5206826964856656.0 / 17592186044416.0;
real Sup_427 = 5206826964856657.0 / 17592186044416.0;
interval ln2_427 = interval(Inf_427,Sup_427);
      // ln2_427 is an optimal inclusion of 427*ln(2);
real rln2_427 = 5206826964856657.0 / 17592186044416.0;
  // rln2_427 is an optimal real approximation for 427*ln(2);
real abs_427 = 2.00699628965226E-016;
  // abs_427 is an abs. error bound of the above real approximation
```

```
real Inf_366 = 8925989082611411.0 / 35184372088832.0;
real Sup_366 = 8925989082611412.0 / 35184372088832.0;
interval ln2_366 = interval(Inf_366,Sup_366);
      // ln2_366 is an optimal inclusion of 366*ln(2);
real rln2_366 = 8925989082611412.0 / 35184372088832.0;
  // rln2_366 is an optimal real approximation for 366*ln(2);
real abs_366 = 1.72028253398765E-016;
  // abs_366 is an abs. error bound of the above real approximation

real Inf_320 = 7804143460206699.0 / 35184372088832.0;
real Sup_320 = 7804143460206700.0 / 35184372088832.0;
interval ln2_320 = interval(Inf_320,Sup_320);
      // ln2_320 is an optimal inclusion of 320*ln(2);
real rln2_320 = 7804143460206699.0 / 35184372088832.0;
  // rln2_320 is an optimal real approximation for 320*ln(2);
real abs_320 = 3.15522446707157E-016;
  // abs_320 is an abs. error bound of the above real approximation

real Inf_259 = 6316478613104797.0 / 35184372088832.0;
real Sup_259 = 6316478613104798.0 / 35184372088832.0;
interval ln2_259 = interval(Inf_259,Sup_259);
      // ln2_259 is an optimal inclusion of 259*ln(2);
real rln2_259 = 6316478613104797.0 / 35184372088832.0;
  // rln2_259 is an optimal real approximation for 259*ln(2);
real abs_259 = 3.44193822273618E-016;
  // abs_259 is an abs. error bound of the above real approximation

real Inf_229 = 5584840163710418.0 / 35184372088832.0;
real Sup_229 = 5584840163710419.0 / 35184372088832.0;
interval ln2_229 = interval(Inf_229,Sup_229);
      // ln2_229 is an optimal inclusion of 229*ln(2);
real rln2_229 = 5584840163710419.0 / 35184372088832.0;
  // rln2_229 is an optimal real approximation for 229*ln(2);
real abs_229 = 5.73564826805304E-016;
  // abs_229 is an abs. error bound of the above real approximation

real Inf_160 = 7804143460206699.0 / 70368744177664.0;
real Sup_160 = 7804143460206700.0 / 70368744177664.0;
interval ln2_160 = interval(Inf_160,Sup_160);
```

```
      // ln2_160 is an optimal inclusion of 160*ln(2);
real rln2_160 = 7804143460206699.0 / 70368744177664.0;
  // rln2_160 is an optimal real approximation for 160*ln(2);
real abs_160 = 1.57761223353579E-016;
  // abs_160 is an abs. error bound of the above real approximation

real Inf_122 = 5950659388407607.0 / 70368744177664.0;
real Sup_122 = 5950659388407608.0 / 70368744177664.0;
interval ln2_122 = interval(Inf_122,Sup_122);
      // ln2_122 is an optimal inclusion of 122*ln(2);
real rln2_122 = 5950659388407608.0 / 70368744177664.0;
  // rln2_122 is an optimal real approximation for 122*ln(2);
real abs_122 = 5.73427511329215E-017;
  // abs_122 is an abs. error bound of the above real approximation

real Inf_61 = 5950659388407607.0 / 140737488355328.0;
real Sup_61 = 5950659388407608.0 / 140737488355328.0;
interval ln2_61 = interval(Inf_61,Sup_61);
      // ln2_61 is an optimal inclusion of 61*ln(2);
real rln2_61 = 5950659388407608.0 / 140737488355328.0;
  // rln2_61 is an optimal real approximation for 61*ln(2);
real abs_61 = 2.86713755664608E-017;
  // abs_61 is an abs. error bound of the above real approximation

real Inf_40 = 7804143460206699.0 / 281474976710656.0;
real Sup_40 = 7804143460206700.0 / 281474976710656.0;
interval ln2_40 = interval(Inf_40,Sup_40);
      // ln2_40 is an optimal inclusion of 40*ln(2);
real rln2_40 = 7804143460206699.0 / 281474976710656.0;
  // rln2_40 is an optimal real approximation for 40*ln(2);
real abs_40 = 3.94403058383947E-017;
  // abs_40 is an abs. error bound of the above real approximation

real Inf_20 = 7804143460206699.0 / 562949953421312.0;
real Sup_20 = 7804143460206700.0 / 562949953421312.0;
interval ln2_20 = interval(Inf_20,Sup_20);
      // ln2_20 is an optimal inclusion of 20*ln(2);
real rln2_20 = 7804143460206699.0 / 562949953421312.0;
  // rln2_20 is an optimal real approximation for 20*ln(2);
real abs_20 = 1.97201529191974E-017;
```

```
  // abs_20 is an abs. error bound of the above real approximation

real T_x(const interval& Xi, const real& delx)
{
    interval Yi=interval(0,Sup(Xi)),A,B;
    real dela,delb,r,eps;

    abs_divh1(Yi,delx,Xi,delx,A,dela); // y/x
    abs_mulh1(A,dela,A,dela,B,delb);   // (y/x)*(y/x)
    abs_lnp1_abs(B,delb,A,dela);

    times2pown(A,-1);    // Division by 2
    times2pown(dela,-1); // Division by 2

    Yi = Xi;  times2pown(Yi,-61);
    abs_ln_abs(Yi,delx,B,delb);
    abs_addh1(A,dela,B,delb,Yi,r);  // [ ... ] ready

    rel_addh1(Yi,r,ln2_61,abs_61,A,eps); // eps: rel. error bound
    return eps;
}

int main()
{
    interval X = interval(comp(0.5,51),comp(0.5,101));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Hinweise zum Programm `lnx2y2_Cj`

1. Die von N abhängigen Konstanten wurden mit dem Programm `Nln2.cpp` von Seite 232 berechnet.

2. Die Funktion `T_x(...)` berechnet für ein Teilintervall `Xi` mit dem Rückgabewert `eps` eine Schranke des relativen Auswertefehlers. Nach dem Quelltext des obigen Programms wird der relative Fehler für das Teilintervall C_{18} mit $N = 61$ berechnet. Für $C_{18} = [2^{50}, 2^{100})$ erhält man den maximalen Auswertefehler $\varepsilon(f, N = 61) = 4.296234 \cdot 10^{-16}$.

Im Falle $\widetilde{y} \leq \widetilde{x} \cdot 2^{-24}$ muss jetzt noch nach (8.18) von Seit 253 der relative Auswertefehler $\varepsilon(g)$ berechnet werden. Zusammen mit $\varepsilon(app)$ von Seite 253 ergibt sich dann der relative Gesamtfehler bei Auswertung von (8.18) nach

$$\varepsilon(f) = \varepsilon(app) + \varepsilon(g) \cdot [1 + \varepsilon(app)]$$

Die Berechnung der N-Werte in den einzelnen C_j erfolgt wieder wie auf Seite 232 beschrieben. Die Ergebnisse sind in der folgenden Tabelle zusammengestellt.

j	C_j	Rel. Fehlerschranke	N	Abs. Schranke bez. $N \cdot \ln(2)$
0	$[2^{950}, 2^{1024})$	$2.391688 \cdot 10^{-16}$	976	$4.58742009063372 \cdot 10^{-16}$
1	$[2^{900}, 2^{950})$	$2.403729 \cdot 10^{-16}$	945	$4.87688015582011 \cdot 10^{-16}$
2	$[2^{850}, 2^{900})$	$2.394524 \cdot 10^{-16}$	885	$1.34782928257584 \cdot 10^{-15}$
3	$[2^{800}, 2^{850})$	$2.401757 \cdot 10^{-16}$	825	$3.18334658073368 \cdot 10^{-15}$
4	$[2^{750}, 2^{800})$	$2.413854 \cdot 10^{-16}$	763	$1.29048653144292 \cdot 10^{-15}$
5	$[2^{700}, 2^{750})$	$2.411218 \cdot 10^{-16}$	732	$3.44056506797529 \cdot 10^{-16}$
6	$[2^{650}, 2^{700})$	$2.388165 \cdot 10^{-16}$	671	$3.15385131231069 \cdot 10^{-16}$
7	$[2^{600}, 2^{650})$	$2.450433 \cdot 10^{-16}$	610	$2.86713755664608 \cdot 10^{-16}$
8	$[2^{550}, 2^{600})$	$2.523129 \cdot 10^{-16}$	549	$2.58042380098147 \cdot 10^{-16}$
9	$[2^{500}, 2^{550})$	$2.460801 \cdot 10^{-16}$	518	$6.88387644547236 \cdot 10^{-16}$
10	$[2^{450}, 2^{500})$	$2.533000 \cdot 10^{-16}$	488	$2.29371004531686 \cdot 10^{-16}$
11	$[2^{400}, 2^{450})$	$2.490190 \cdot 10^{-16}$	427	$2.00699628965226 \cdot 10^{-16}$
12	$[2^{350}, 2^{400})$	$2.549755 \cdot 10^{-16}$	366	$1.72028253398765 \cdot 10^{-16}$
13	$[2^{300}, 2^{350})$	$2.570860 \cdot 10^{-16}$	320	$3.15522446707157 \cdot 10^{-16}$
14	$[2^{250}, 2^{300})$	$2.741281 \cdot 10^{-16}$	259	$3.44193822273618 \cdot 10^{-16}$
15	$[2^{200}, 2^{250})$	$2.816229 \cdot 10^{-16}$	229	$5.73564826805304 \cdot 10^{-16}$
16	$[2^{150}, 2^{200})$	$2.990636 \cdot 10^{-16}$	160	$1.57761223353579 \cdot 10^{-16}$
17	$[2^{100}, 2^{150})$	$3.131749 \cdot 10^{-16}$	122	$5.73427511329215 \cdot 10^{-17}$
18	$[2^{50}, 2^{100})$	$3.883653 \cdot 10^{-16}$	61	$2.86713755664608 \cdot 10^{-17}$
19	$[2^{30}, 2^{50})$	$4.073594 \cdot 10^{-16}$	40	$3.94403058383947 \cdot 10^{-17}$
20	$[2^{20}, 2^{30})$	$4.491234 \cdot 10^{-16}$	20	$1.97201529191974 \cdot 10^{-17}$

Die relativen Fehlerschranken aus Spalte 3 der obigen Tabelle wurden für jedes C_j mit dem folgenden Programm `lnx2y2_Cj2.cpp` berechnet:

```
// Program lnx2y2_Cj2.cpp for calculating the relative error by
// evaluating the function    g(x) = N*ln(2) + ln(2^(-N)*x)
// for 2^10 <= x <= MaxReal.
// The approximation error eps(app) is considered and eps(f) is
// evaluated by:         eps(f) = eps(app) + eps(g)[1+eps(app)]

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <imath.hpp>
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real Inf_976 = 5950659388407607.0 / 8796093022208.0;
real Sup_976 = 5950659388407608.0 / 8796093022208.0;
interval ln2_976 = interval(Inf_976,Sup_976);
      // ln2_976 is an optimal inclusion of 976*ln(2);
real rln2_976 = 5950659388407608.0 / 8796093022208.0;
  // rln2_976 is an optimal real approximation for 976*ln(2);
real abs_976 = 4.58742009063372E-016;
  // abs_976 is an abs. error bound of the above real approximation

real Inf_945 = 5761652788980727.0 / 8796093022208.0;
real Sup_945 = 5761652788980728.0 / 8796093022208.0;
interval ln2_945 = interval(Inf_945,Sup_945);
      // ln2_945 is an optimal inclusion of 945*ln(2);
real rln2_945 = 5761652788980727.0 / 8796093022208.0;
  // rln2_945 is an optimal real approximation for 945*ln(2);
real abs_945 = 4.87688015582011E-016;
  // abs_945 is an abs. error bound of the above real approximation

real Inf_885 = 5395833564283537.0 / 8796093022208.0;
real Sup_885 = 5395833564283538.0 / 8796093022208.0;
interval ln2_885 = interval(Inf_885,Sup_885);
      // ln2_885 is an optimal inclusion of 885*ln(2);
real rln2_885 = 5395833564283538.0 / 8796093022208.0;
  // rln2_885 is an optimal real approximation for 885*ln(2);
real abs_885 = 1.34782928257584E-015;
```

```
  // abs_885 is an abs. error bound of the above real approximation

real Inf_825 = 5030014339586348.0 / 8796093022208.0;
real Sup_825 = 5030014339586349.0 / 8796093022208.0;
interval ln2_825 = interval(Inf_825,Sup_825);
      // ln2_825 is an optimal inclusion of 825*ln(2);
real rln2_825 = 5030014339586349.0 / 8796093022208.0;
  // rln2_825 is an optimal real approximation for 825*ln(2);
real abs_825 = 3.18334658073368E-015;
  // abs_825 is an abs. error bound of the above real approximation

real Inf_763 = 4652001140732586.0 / 8796093022208.0;
real Sup_763 = 4652001140732587.0 / 8796093022208.0;
interval ln2_763 = interval(Inf_763,Sup_763);
      // ln2_763 is an optimal inclusion of 763*ln(2);
real rln2_763 = 4652001140732587.0 / 8796093022208.0;
  // rln2_763 is an optimal real approximation for 763*ln(2);
real abs_763 = 1.29048653144292E-015;
  // abs_763 is an abs. error bound of the above real approximation

real Inf_732 = 4652001140732586.0 / 8796093022208.0;
real Sup_732 = 4652001140732587.0 / 8796093022208.0;
interval ln2_732 = interval(Inf_732,Sup_732);
      // ln2_732 is an optimal inclusion of 732*ln(2);
real rln2_732 = 4652001140732587.0 / 8796093022208.0;
  // rln2_732 is an optimal real approximation for 732*ln(2);
real abs_732 = 1.29048653144292E-015;
  // abs_732 is an abs. error bound of the above real approximation

real Inf_671 = 8182156659060460.0 / 17592186044416.0;
real Sup_671 = 8182156659060461.0 / 17592186044416.0;
interval ln2_671 = interval(Inf_671,Sup_671);
      // ln2_671 is an optimal inclusion of 671*ln(2);
real rln2_671 = 8182156659060461.0 / 17592186044416.0;
  // rln2_671 is an optimal real approximation for 671*ln(2);
real abs_671 = 3.15385131231069E-016;
  // abs_671 is an abs. error bound of the above real approximation

real Inf_610 = 7438324235509509.0 / 17592186044416.0;
```

```
real Sup_610 = 7438324235509510.0 / 17592186044416.0;
interval ln2_610 = interval(Inf_610,Sup_610);
      // ln2_610 is an optimal inclusion of 610*ln(2);
real rln2_610 = 7438324235509510.0 / 17592186044416.0;
  // rln2_610 is an optimal real approximation for 610*ln(2);
real abs_610 = 2.86713756E-016;
  // abs_610 is an abs. error bound of the above real approximation

real Inf_549 = 6694491811958558.0 / 17592186044416.0;
real Sup_549 = 6694491811958559.0 / 17592186044416.0;
interval ln2_549 = interval(Inf_549,Sup_549);
      // ln2_549 is an optimal inclusion of 549*ln(2);
real rln2_549 = 6694491811958559.0 / 17592186044416.0;
  // rln2_549 is an optimal real approximation for 549*ln(2);
real abs_549 = 2.58042381E-016;
  // abs_549 is an abs. error bound of the above real approximation

real Inf_518 = 6316478613104797.0 / 17592186044416.0;
real Sup_518 = 6316478613104798.0 / 17592186044416.0;
interval ln2_518 = interval(Inf_518,Sup_518);
      // ln2_518 is an optimal inclusion of 518*ln(2);
real rln2_518 = 6316478613104797.0 / 17592186044416.0;
  // rln2_518 is an optimal real approximation for 518*ln(2);
real abs_518 = 6.88387645E-016;
  // abs_518 is an abs. error bound of the above real approximation

real Inf_488 = 5950659388407607.0 / 17592186044416.0;
real Sup_488 = 5950659388407608.0 / 17592186044416.0;
interval ln2_488 = interval(Inf_488,Sup_488);
      // ln2_488 is an optimal inclusion of 488*ln(2);
real rln2_488 = 5950659388407608.0 / 17592186044416.0;
  // rln2_488 is an optimal real approximation for 488*ln(2);
real abs_488 = 2.29371004531686E-016;
  // abs_488 is an abs. error bound of the above real approximation

real Inf_427 = 5206826964856656.0 / 17592186044416.0;
real Sup_427 = 5206826964856657.0 / 17592186044416.0;
interval ln2_427 = interval(Inf_427,Sup_427);
      // ln2_427 is an optimal inclusion of 427*ln(2);
real rln2_427 = 5206826964856657.0 / 17592186044416.0;
```

```
  // rln2_427 is an optimal real approximation for 427*ln(2);
real abs_427 = 2.00699628965226E-016;
  // abs_427 is an abs. error bound of the above real approximation

real Inf_366 = 8925989082611411.0 / 35184372088832.0;
real Sup_366 = 8925989082611412.0 / 35184372088832.0;
interval ln2_366 = interval(Inf_366,Sup_366);
      // ln2_366 is an optimal inclusion of 366*ln(2);
real rln2_366 = 8925989082611412.0 / 35184372088832.0;
  // rln2_366 is an optimal real approximation for 366*ln(2);
real abs_366 = 1.72028253398765E-016;
  // abs_366 is an abs. error bound of the above real approximation

real Inf_320 = 7804143460206699.0 / 35184372088832.0;
real Sup_320 = 7804143460206700.0 / 35184372088832.0;
interval ln2_320 = interval(Inf_320,Sup_320);
      // ln2_320 is an optimal inclusion of 320*ln(2);
real rln2_320 = 7804143460206699.0 / 35184372088832.0;
  // rln2_320 is an optimal real approximation for 320*ln(2);
real abs_320 = 3.15522446707157E-016;
  // abs_320 is an abs. error bound of the above real approximation

real Inf_259 = 6316478613104797.0 / 35184372088832.0;
real Sup_259 = 6316478613104798.0 / 35184372088832.0;
interval ln2_259 = interval(Inf_259,Sup_259);
      // ln2_259 is an optimal inclusion of 259*ln(2);
real rln2_259 = 6316478613104797.0 / 35184372088832.0;
  // rln2_259 is an optimal real approximation for 259*ln(2);
real abs_259 = 3.44193822273618E-016;
  // abs_259 is an abs. error bound of the above real approximation

real Inf_229 = 5584840163710418.0 / 35184372088832.0;
real Sup_229 = 5584840163710419.0 / 35184372088832.0;
interval ln2_229 = interval(Inf_229,Sup_229);
      // ln2_229 is an optimal inclusion of 229*ln(2);
real rln2_229 = 5584840163710419.0 / 35184372088832.0;
  // rln2_229 is an optimal real approximation for 229*ln(2);
real abs_229 = 5.73564826805304E-016;
  // abs_229 is an abs. error bound of the above real approximation
```

```
real Inf_160 = 7804143460206699.0 / 70368744177664.0;
real Sup_160 = 7804143460206700.0 / 70368744177664.0;
interval ln2_160 = interval(Inf_160,Sup_160);
      // ln2_160 is an optimal inclusion of 160*ln(2);
real rln2_160 = 7804143460206699.0 / 70368744177664.0;
  // rln2_160 is an optimal real approximation for 160*ln(2);
real abs_160 = 1.57761223353579E-016;
  // abs_160 is an abs. error bound of the above real approximation

real Inf_122 = 5950659388407607.0 / 70368744177664.0;
real Sup_122 = 5950659388407608.0 / 70368744177664.0;
interval ln2_122 = interval(Inf_122,Sup_122);
      // ln2_122 is an optimal inclusion of 122*ln(2);
real rln2_122 = 5950659388407608.0 / 70368744177664.0;
  // rln2_122 is an optimal real approximation for 122*ln(2);
real abs_122 = 5.73427511329215E-017;
  // abs_122 is an abs. error bound of the above real approximation

real Inf_61 = 5950659388407607.0 / 140737488355328.0;
real Sup_61 = 5950659388407608.0 / 140737488355328.0;
interval ln2_61 = interval(Inf_61,Sup_61);
      // ln2_61 is an optimal inclusion of 61*ln(2);
real rln2_61 = 5950659388407608.0 / 140737488355328.0;
  // rln2_61 is an optimal real approximation for 61*ln(2);
real abs_61 = 2.86713755664608E-017;
  // abs_61 is an abs. error bound of the above real approximation

real Inf_40 = 7804143460206699.0 / 281474976710656.0;
real Sup_40 = 7804143460206700.0 / 281474976710656.0;
interval ln2_40 = interval(Inf_40,Sup_40);
      // ln2_40 is an optimal inclusion of 40*ln(2);
real rln2_40 = 7804143460206699.0 / 281474976710656.0;
  // rln2_40 is an optimal real approximation for 40*ln(2);
real abs_40 = 3.94403058383947E-017;
  // abs_40 is an abs. error bound of the above real approximation

real Inf_20 = 7804143460206699.0 / 562949953421312.0;
real Sup_20 = 7804143460206700.0 / 562949953421312.0;
interval ln2_20 = interval(Inf_20,Sup_20);
```

```
      // ln2_20 is an optimal inclusion of 20*ln(2);
real rln2_20 = 7804143460206699.0 / 562949953421312.0;
  // rln2_20 is an optimal real approximation for 20*ln(2);
real abs_20 = 1.97201529191974E-017;
  // abs_20 is an abs. error bound of the above real approximation

const real app1 = comp(0.5,-48); // app1 = 2^(-49)

real T_x(const interval& Xi, const real& delx)
// Calculation of eps(g) with g(x) := ln(2^(-N)*x)] + N*ln(2) = ln(x)
{
    interval Yi = Xi, A, B;
    real dela,eps;
    times2pown(Yi,-20);
    abs_ln_abs(Yi,delx,A,dela);  // ln(2^(-N)*a)
    rel_addh1(A,dela,ln2_20,abs_20,B,eps); // eps: rel. error bound
    return eps;
}

int main()
{
    interval X = interval(comp(0.5,21),comp(0.5,31));// X=[2^20,2^30]
    real r,Sa,bnd,diam = 1e-5,delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    // bnd: Relat. error bound for evaluating ln(2^(-N)*x)] + N*ln(2)

    // Now calculating the approximation error:
    interval app2 = ln( interval(Inf(X)) ),app;
    app = app1 / app2; // app: inclusion of eps(app)
    Sa = Sup(app); // eps(f) = eps(app) + eps(g)[1+eps(app)]:
    r = addu(1,Sa);
    r = mulu(r,bnd);
    bnd = addu(r,Sa); // bnd = eps(f)
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Hinweise zum Programm `lnx2y2_Cj2`

1. Die von N abhängigen Konstanten wurden mit dem Programm `Nln2.cpp` von Seite 232 berechnet.

2. Die Funktion `T_x(...)` berechnet für ein Teilintervall `Xi` mit dem Rückgabewert `eps` eine Schranke des relativen Auswertefehlers. Nach dem Quelltext des obigen Programms wird der relative Fehler für das Teilintervall C_{20} mit $N = 20$ berechnet. Für $C_{20} = [2^{20}, 2^{30})$ erhält man den maximalen Auswertefehler $\varepsilon(f, N = 20) = 4.491234 \cdot 10^{-16}$.

Für C_{20} liefert das Programm `lnx2y2_Cj2.cpp` mit $N = 20$ die maximale Fehlerschranke:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_7(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, N = 20) = 4.491234 \cdot 10^{-16}$$

Zusammenfassung:

Für alle $\widetilde{x} \in [2^{20}, 2^{1024})$ und damit für alle $s := \widetilde{x}^2 + \widetilde{y}^2 \in A_7 = [2^{40}, 2^{2049})$ gilt bei Auswertung von (8.17) bzw. (8.18) auf Seite 253 für die Schranke $\varepsilon(f, A_7)$ des relativen Fehlers:

$$\boxed{\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_7(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, A_7) = 4.491234 \cdot 10^{-16}} \tag{8.19}$$

Die Fehlerabschätzung muss jetzt noch für die Quadratsummen $s := \widetilde{x}^2 + \widetilde{y}^2$ in folgendem Intervall durchgeführt werden:

$$2^{-39} \leq s < 2^{40}, \quad s := \widetilde{x}^2 + \widetilde{y}^2, \quad 0 \leq \widetilde{y} \leq \widetilde{x}; \tag{8.20}$$

Beachten Sie bitte, dass die Berechnung von s jetzt weder zum Underflow noch zum Overflow führen kann, wenn s im Programm `lnx2y2` auf Seite 288 im Akkumulator `dot` rundungsfehlerfrei berechnet und mit der Anweisung `s=rnd(dot)` zur nächsten *double*-Zahl `s` $\in S(2, 53)$ gerundet wird. Die Schranke $\varepsilon(s)$ des dabei auftretenden relativen Fehlers ε_s ist dann gegeben durch:

$$\frac{s - \mathtt{s}}{s} = \varepsilon_s, \quad |\varepsilon_s| \leq \varepsilon(s) := 2^{-53} = 1.1102\ldots \cdot 10^{-16}; \tag{8.21}$$

Beachten Sie bitte, dass die ins *double*-Format gerundete Quadratsumme $s := \widetilde{x}^2 + \widetilde{y}^2$ im Programmquelltext mit `s`, bei den nachfolgenden Fehlerabschätzungen jedoch mit $\widetilde{s}$ bezeichnet wird.

Fehlerabschätzung für $\widetilde{x}^2+\widetilde{y}^2 \in A_4 = [1-\frac{11}{64}, 1+\frac{11}{64}]$

Man könnte jetzt annehmen, dass die Auswertung unserer Funktion

$$f(x,y) := \ln(\sqrt{x^2+y^2}) \equiv \frac{1}{2}\ln(x^2+y^2)$$

mit dem rechten Term $\frac{1}{2}\ln(x^2+y^2)$ wegen der vergleichsweise kleinen Fehlerschranke $\varepsilon(s) = 2^{-53}$ jetzt problemlos möglich ist. Im Falle $s \approx 1$ ist dies jedoch ein Irrtum, da wir dann die Logarithmus-Funktion in der Näher ihrer Nullstelle mit fehlerbehafteten Argumenten aufrufen und damit erhebliche relative Fehler erzeugen. Diese Schwierigkeiten umgeht man bei Anwendung der Identität

$$f(x,y) := \ln(\sqrt{x^2+y^2}) \equiv \frac{1}{2}\cdot\ln(1+[x^2+y^2-1]) =: f_4(x,y), \tag{8.22}$$

wobei man $r := x^2+y^2-1$ im Akkumulator rundungsfehlerfrei berechnet und die rechte Seite von (8.22) mit Hilfe der Funktion `lnp1(r)` $\approx \ln(1+$ `r`$)$ auswertet. Mit `r` $= \widetilde{r}$ bezeichnen wir den aus dem Akku ausgelesenen Maschinenwert, wobei gilt:

$$\widetilde{r} = r\cdot(1+\varepsilon_r), \quad \text{mit: } |\varepsilon_r| \le 2^{-53} = \varepsilon(r)$$

Mit $\varepsilon(r) = 2^{-53}$ setzen wir voraus, dass $\widetilde{r}$ und r beide entweder verschwinden oder beide im normalisierten Bereich liegen. Für $\widetilde{x} = 1$ und $\widetilde{y} \to 0$ ist diese Voraussetzung jedoch nicht erfüllt, da $r = \widetilde{y}^2$ jetzt durchaus in den denormalisierten Bereich fallen kann. Wir betrachten daher zunächst den Fall:

$\widetilde{x} = 1,\ 0 \le \widetilde{y} < 2^{-28}$;

Mit $r = y^2$ und wegen $\widetilde{y} < 2^{-28}$ erhalten wir

$$f(x,y) := \ln(\sqrt{x^2+y^2}) \equiv \frac{1}{2}\cdot\ln(1+y^2) \approx \frac{1}{2}\cdot y^2 =: f_{4a}(y), \tag{8.23}$$

und $f_{4a}(\widetilde{y})$ kann jetzt für $\widetilde{y} \to 0$ in den Underflow-Bereich fallen. In diesem Fall ist dann nur eine Schranke des absoluten Fehlers $|f_4(1,\widetilde{y}) - \widetilde{f}_{4a}(\widetilde{y})|$ zu berechnen, wobei das Maschinenergebnis $\widetilde{f}_{4a}(\widetilde{y})$ wie folgt definiert wird:

$$\widetilde{f}_{4a}(\widetilde{y}) := (0.5 \odot \widetilde{y}) \odot \widetilde{y} \tag{8.24}$$

Um entscheiden zu können, für welche $\widetilde{y} \in S(2,53)$ die Maschinenwerte $\widetilde{f}_{4a}(\widetilde{y})$ in den denormalisierten Bereich fallen können, stellen wir folgende Frage:
Für welches $y_0 \in S(2,53)$ gilt:

1. $\widetilde{y} < y_0 \implies \widetilde{f}_{4a}(\widetilde{y}) = (0.5 \odot \widetilde{y}) \odot \widetilde{y} <$ `MinReal` $= 2^{-1022}$, falls bei beiden Multiplikationen abgerundet wird.
2. $\widetilde{y} \ge y_0 \implies \widetilde{f}_{4a}(\widetilde{y}) = (0.5 \odot \widetilde{y}) \odot \widetilde{y} \ge$ `MinReal` $= 2^{-1022}$, falls die beiden Multiplikationen im beliebigen Rundungsmodus erfolgen.

Mit dem folgenden Programm `lnx2y2_y0.cpp`

```
// Program  lnx2y2_y0.cpp  for calculating the machine number yo, so
// that the following statements are valid for the machine values y:
// 1. y <  yo ==>  g(y):= (0.5*y)*y < MinReal with rounding downwards
// 2. y >= y0 ==>  g(y):= (0.5*y)*y >= MinReal with arbitrary ronding
//                                    modus by the two multiplications
#include <imath.hpp>    // For z = sqrt( interval(2) )
#include "bnd_util.hpp" // For IEEE2frac_int(b0,Z,N);
#include <iostream>     // For cout
using namespace cxsc;
using namespace std;

interval z = sqrt( interval(2) )*comp(0.5,-510);
real b0 = Sup(z),f4a,Z,N;

int main()
{
    do {
        b0 = pred(b0);
        f4a = muld(0.5,b0);   // rouding downwards
        f4a = muld(f4a,b0);   // rouding downwards
    } while(f4a >= MinReal);

    b0 = succ(b0);  // This b0 is the searched y0
    IEEE2frac_int(b0,Z,N);
    cout << Fixed << SetPrecision(1,1) << "y0 = "
         << Z << " / " << N << endl;
        f4a = muld(0.5,b0);
        f4a = muld(f4a,b0);
        if (f4a >= MinReal) cout << "f4a >= MinReal" << endl;
        else cout << "f4a < MinReal" << endl;
}
```

erhält man für die Maschinenzahl y_0 das Ergebnis:

$$
\begin{aligned}
y_0 \;=\;& 6369051672525773.0/ \\
& 30191699398572330817932436647906151127335369763 33 \\
& 15234270096504019649932991371908166890138014212 70 \\
& 14033174700024611075919816467703939834106049147 40 \\
& 1146156834919516261580 8.0 \approx \sqrt{2}\cdot 2^{-511}
\end{aligned}
$$

Für $\widetilde{x} = 1$ und $0 \leq \widetilde{y} \leq y_0$ berechnen wir jetzt eine Schranke des absoluten Auswertefehlers $|\widetilde{f}_{4a} - \frac{1}{2}\widetilde{y}^2|$. Mit dem Programm `lnx2y2_abs`

```
// Program lnx2y2_abs.cpp for calculating the absolute error by
// evaluating the term f4a(b) := (0.5*b)*b;
// for 0 <= b <= b0;

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

const real b0 = 6369051672525773.0 /
                30191699398572330817932436647906151127335369763331523
                42700965040196499329913719081668901380142127014033174
                70002461107591981646770393983410604914740114615683491
                95162615808.0;

real T_x(const interval& Xi, const real& delx)
{
    interval A=interval(0.5),B; // Including the factor 1/2
    real absa,absb;
    abs_mulh1(A,0,Xi,delx,B,absb);    // absb: abs. error for 0.5*Xi
    abs_mulh1(B,absb,Xi,delx,A,absa); // absa: abs. error for the
    return absa;                      // evaluation of (0.5*Xi)*Xi
}

int main()
{
    interval X = interval(0,b0);
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Absolute error bound = " << bnd << endl;
}
```

erhält man das Ergebnis:

$$|\widetilde{f}_{4a} - \frac{1}{2}\widetilde{y}^2| = |(0.5 \odot \widetilde{y}) \odot \widetilde{y} - \frac{1}{2}\widetilde{y}^2| \leq 2.225074 \cdot 10^{-308} = \Delta(f_{4a})$$

Wir benötigen jetzt noch eine Oberschranke des absoluten Approximationsfehlers $|\frac{1}{2}\cdot\ln(1+y^2)-\frac{1}{2}\cdot y^2|$, ebenfalls für $0\le\widetilde{y}\le y_0$. Da $\ln(1+y^2)$ eine alternierende Leibniz-Reihe besitzt, gilt:

$$\left|\frac{1}{2}\cdot\ln(1+y^2)-\frac{1}{2}\cdot y^2\right|\le\frac{1}{2}\cdot\frac{1}{2}\cdot y^4\le\frac{1}{4}\cdot y_0^4<2^{-2043}\sim 10^{-615}$$

Nach (7.2) von Seite 186 folgt dann für den absoluten Gesamtfehler

$$\left|\frac{1}{2}\cdot\ln(1+\widetilde{y}^2)-\widetilde{f}_{4a}(\widetilde{y})\right|\le\Delta(f_{4a})+2^{-2043}<2.225075\cdot10^{-308}=\Delta(f_4)$$

Zusammenfassung: Für $\widetilde{x}=1$ und $0\le\widetilde{y}\le y_0$ gilt mit $f_{4a}(y):=\frac{1}{2}\cdot y^2$

(8.25)
$$\boxed{\left|\ln(\sqrt{1+\widetilde{y}^2})-\widetilde{f}_{4a}(\widetilde{y})\right|\le\Delta(f_4)=2.225075\cdot10^{-308}}$$

Für $\widetilde{x}=1$ bleibt jetzt noch der Restbereich $y_0\le\widetilde{y}<2^{-28}$, in dem $f_{4a}(\widetilde{y}):=\frac{1}{2}\cdot\widetilde{y}^2$ und $\widetilde{f}_{4a}(\widetilde{y}):=(0.5\odot\widetilde{y})\odot\widetilde{y}$ nach Seite 269 bei beliebigem Rundungsmodus beide im normalisierten Zahlenbereich liegen, so dass jetzt wieder eine Schranke des relativen Fehlers berechnet werden kann. Wegen $0.5\odot\widetilde{y}=0.5\cdot\widetilde{y}$ ist der relative Auswertefehler bez. $\widetilde{f}_{4a}(\widetilde{y}):=(0.5\odot\widetilde{y})\odot\widetilde{y}=(0.5\cdot\widetilde{y})\odot\widetilde{y}$ nur durch die letzte Multiplikation gegeben, und es gilt bei vorausgesetzter hochgenauer Arithmetik $\varepsilon(\odot)=2^{-52}=2.220446\ldots\cdot10^{-16}$. Der relative Approximationsfehler lässt sich wie folgt abschätzen:

$$\begin{aligned}\left|\frac{\frac{1}{2}\cdot\ln(1+y^2)-\frac{1}{2}\cdot y^2}{\frac{1}{2}\cdot\ln(1+y^2)}\right| &= \frac{\left|\ln(1+y^2)-y^2\right|}{\ln(1+y^2)}\le\frac{\frac{1}{2}\,y^4}{\ln(1+y^2)}\le\frac{2^{-113}}{\ln(1+2^{-56})}\\ &< 6.938894\cdot10^{-18}=\varepsilon(app),\end{aligned}$$

wobei der letzte Bruch durch Intervallrechnung eingeschlossen wurde und $\varepsilon(app)$ die berechnete Intervallobergrenze ist. Mit $\varepsilon(f_{4a})=\varepsilon(app)+\varepsilon(\odot)\cdot[1+\varepsilon(app)]$ folgt dann nach (7.1) auf Seite 186 für die Schranke $\varepsilon(f_{4a})$ des relativen Gesamtfehlers

$$\varepsilon(f_{4a})=2.289836\cdot10^{-16}$$

Zusammenfassung: Für $\widetilde{x}=1$ und $y_0\le\widetilde{y}<2^{-28}$ gilt mit $f_{4a}(y):=\frac{1}{2}\cdot y^2$

(8.26)
$$\boxed{\left|\frac{\ln(\sqrt{1+\widetilde{y}^2})-\widetilde{f}_{4a}(\widetilde{y})}{\ln(\sqrt{1+\widetilde{y}^2})}\right|\le\varepsilon(f_{4a})=2.289836\cdot10^{-16}}$$

Mit den Ergebnissen von Seite 272 fehlt jetzt noch die Fehlerabschätzung im Bereich $s := \widetilde{x}^2 + \widetilde{y}^2 \in A_4 = [1 - \frac{11}{64}, 1 + \frac{11}{64}]$, wobei $(\widetilde{x} \neq 1 \vee \widetilde{y} \geq y_0)$ erfüllt sein muss. Nach (8.22) von Seite 269 erfolgt die Auswertung von $f(x,y) := \ln(\sqrt{x^2+y^2})$ in diesem Bereich durch

$$f(x,y) \equiv \frac{1}{2} \cdot \ln(1 + [x^2 + y^2 - 1]) =: f_4(x,y),$$

wobei $r := x^2+y^2-1$ im Akkumulator exakt berechnet wird. Zur Fehlerabschätzung benötigen wir sowohl die relative Fehlerschranke, die beim Auslesen des Akkumulators ins *double*-Format entsteht, als auch eine Einschließung aller exakten r-Werte. Zur Berechnung dieser Einschließung ist zu beachten, dass im Programm `lnx2y2` von Seite 288 $s := x^2 + y^2$ ebenfalls im Akku exakt berechnet und mit `s = rnd(dot)` ins *double*-Format gerundet wird. Es gilt daher

$$\widetilde{s} = \mathtt{s} = s \cdot (1 + \varepsilon_s), \quad \text{mit} \quad |\varepsilon_s| \leq \varepsilon(s) = 2^{-53}$$

Bevor $r = x^2 + y^2 - 1$ im Akku berechnet wird, erfolgt im Programm `lnx2y2` die Abfrage: `if (s >= 0.828125 && s <= 1.171875) {...}`, daraus ergeben sich die folgenden Einschließungen:

$$\begin{aligned}
& 1 - \frac{11}{64} \leq \widetilde{s} \leq 1 + \frac{11}{64} \quad \Longleftrightarrow \quad 1 - \frac{11}{64} \leq s \cdot (1+\varepsilon_s) \leq 1 + \frac{11}{64} \\
\Longleftrightarrow \quad & \frac{1 - \frac{11}{64}}{1+\varepsilon_s} \leq s \leq \frac{1 + \frac{11}{64}}{1+\varepsilon_s} \quad \Longleftrightarrow \quad \frac{1 - \frac{11}{64}}{1+\varepsilon_s} - 1 \leq r \leq \frac{1 + \frac{11}{64}}{1+\varepsilon_s} - 1 \\
\Longleftrightarrow \quad & \frac{-\frac{11}{64} - \varepsilon_s}{1+\varepsilon_s} \leq r \leq \frac{\frac{11}{64} - \varepsilon_s}{1+\varepsilon_s} \\
\Longrightarrow \quad & \frac{-\frac{11}{64} - \varepsilon(s)}{1+\varepsilon(s)} \leq r \leq \frac{\frac{11}{64} + \varepsilon(s)}{1-\varepsilon(s)}, \quad \varepsilon(s) := 2^{-53}; \qquad (8.27)
\end{aligned}$$

Wertet man die Brüche in (8.27) intervallmäßig aus, so erhält man für r die Einschließung:

$$-0.17187501 \leq r \leq +0.17187501 \qquad (8.28)$$

Im Programm `lnx2y2` gilt `dot` $= r$, und durch `r = rnd(dot)` wird der exakte r-Wert in die *double*-Variable `r` $= \widetilde{r}$ ausgelesen. Für den dabei auftretenden relativen Fehler ε_r gilt:

$$\widetilde{r} = r \cdot (1 + \varepsilon_r), \quad |\varepsilon_r| \leq \varepsilon(r) = 2^{-53}; \qquad (8.29)$$

Mit $\varepsilon(r) = 2^{-53}$ setzen wir dabei voraus, dass r und $\widetilde{r}$ stets im normalisierten Zahlenbereich liegen. Würde r in den denormalisierten Bereich fallen[6], so müsste man

[6] Aus $r = 0$ folgt $\widetilde{r} = 0$, und der relative Fehler kann Null gesetzt werden.

analog zur Seite 16 für $\varepsilon(r)$ sehr viel größere Schranken wählen! Von der Anschauung her ist es jedoch keinesfalls sicher, dass mit $r := \widetilde{x}^2 + \widetilde{y}^2 - 1$ und $|r| > 0$ stets gelten soll: $|r| > 2^{-1022}$. Die Frage kann auch so gestellt werden: Warum sollte es in der $(\widetilde{x}, \widetilde{y})$-Ebene nicht Gitterpunkte $P(\widetilde{x}|\widetilde{y})$, mit $\widetilde{x}, \widetilde{y} \in S(2, 53)$ geben, die so dicht am Einheitskreis liegen, dass gilt: $0 < |r| < 2^{-1022}$? Zur Beantwortung der Frage formulieren wir den

Satz 8.2.1 Unter den Voraussetzungen

1. $\frac{1}{2}\sqrt{2} < \widetilde{x}$ und $\widetilde{x} \neq 1$
2. $0 \leq \widetilde{y} \leq \widetilde{x}; \quad \widetilde{x}, \widetilde{y} \in S(2, 53)$
3. $r := \widetilde{x}^2 + \widetilde{y}^2 - 1 \neq 0$

gilt: $|\widetilde{x}^2 + \widetilde{y}^2 - 1| \geq 2^{-158} = 2.73691106\ldots \cdot 10^{-48}$

Zum Beweis betrachten wir zunächst den Fall $\widetilde{x} > 1$. Dann ist $\widetilde{x}^2 - 1 > 0$ und $\widetilde{y}^2 \geq 0$, und $r > 0$ wird dann minimal für $\widetilde{y} = 0$ und $\widetilde{x} = \texttt{succ}(1)$, d.h. nach (1.8) von Seite 8 gilt: $r \geq \texttt{succ}^2(1) - 1 + 0 = (1 + 2^{-52})^2 - 1 = 2^{-51} + 2^{-104} \geq 2^{-51} = 4.4408\ldots \cdot 10^{-16}$, womit obige Behauptung erfüllt ist. Die Voraussetzung 1. reduziert sich damit auf die Bedingung: $\frac{1}{2}\sqrt{2} < \widetilde{x} < 1$. Damit ist jetzt $\widetilde{x}^2 - 1 < 0$ und $\widetilde{y}^2 \geq 0$, und $|r| > 0$ wird minimal, wenn gilt:

a. $1 - \widetilde{x}^2 > 0$ wird minimal

b. Die nach Voraussetzung 3. verschiedenen positiven Werte $1 - \widetilde{x}^2$ und $\widetilde{y}^2$ unterscheiden sich minimal.

Alle benachbarten Rasterzahlen $\widetilde{x}$ haben wegen $\frac{1}{2}\sqrt{2} < \widetilde{x} < 1$ nach (1.7) von Seite 8 den Abstand $\tau = 2^{-53}$, d.h. es gilt: $\widetilde{x} = 1 - N \cdot \tau$, mit $N \in \mathbb{N}$ und damit:

$$1 - \widetilde{x}^2 = 1 - (1 - N \cdot \tau)^2 = N \cdot \tau \cdot (2 - N \cdot \tau)$$

Danach wird $1 - \widetilde{x}^2$ für die in $\frac{1}{2}\sqrt{2} < \widetilde{x} < 1$ liegenden Maschinenzahlen $\widetilde{x}$ minimal für $N = 1$, und es gilt:

$$1 - \widetilde{x}^2 \geq 2\tau - \tau^2 = 2^{-52} - 2^{-106} = 0.\underbrace{111111\ldots111111}_{\text{54 Dualziffern 1}} \cdot 2^{-52} \tag{8.30}$$

Damit haben wir die obige Bedingung a. erfüllt. Um jetzt nach b. die kleinstmögliche Differenz der positiven Zahlen $1 - \widetilde{x}^2$ und $\widetilde{y}^2$ zu ermitteln, fragen wir zunächst nach

der größtmöglichen Mantissenbreite von $1-\widetilde{x}^2$ und $\widetilde{y}^2$. Da $\widetilde{y} \in S(2,53)$ 53 Mantissenstellen besitzt, muss man für $\widetilde{y}^2$ maximal 106 Mantissenstellen bereitstellen, und das nächste Beispiel zeigt, dass man für $1-\widetilde{x}^2$ nur maximal 105 Stellen benötigt[7]:

$$\begin{aligned} 1 &= 1.00000000\ldots000000 \\ \widetilde{x}^2 &= \underline{0.10111111\ldots11} \\ 1-\widetilde{x}^2 &= 0.\underbrace{01000000\ldots01}_{\text{105 Dualziffern}} \end{aligned}$$

Damit besitzt $\widetilde{y}^2$ die maximale Breite von 106 Mantissenziffern, und man erhält den theoretisch kleinstmöglichen positiven Wert für r, wenn man $\widetilde{y}^2$ mit 106 Dualstellen so wählt, dass sich die Bit-Muster von $\widetilde{y}^2$ und $1-\widetilde{x}^2$ nur auf der jeweils letzten Stelle unterscheiden:

$$\begin{aligned} 1-\widetilde{x}^2 &= 0.\overbrace{111111\ldots111111}^{\text{54 Dualziffern}}000\ldots000\cdot 2^{-52} \\ \widetilde{y}^2 &= 0.\underbrace{111111\ldots111111000\ldots001}_{\text{106 Dualziffern}}\cdot 2^{-52} \end{aligned}$$

Die kleinstmögliche positive Differenz $r = \widetilde{x}^2+\widetilde{y}^2-1$ ergibt sich daraus zu

$$r \geq r_0 := 2^{-106}\cdot 2^{-52} = 2^{-158} = 2.73691106\ldots\cdot 10^{-48}$$

Für die größtmögliche negative Differenz erhält man entsprechend $r \leq -2^{-158}$ ∎

Anmerkungen zum Satz 8.2.1:

1. Die berechnete Unterschranke r_0 der positiven Beträge $|r|$ ist der theoretisch kleinstmögliche Wert, der für keinen Gitterpunkt $P(\widetilde{x}|\widetilde{y})$ wirklich angenommen werden muss. Im Bereich $0.9999999 < \widetilde{x} < 1$ habe ich mit einem **C-XSC** Programm den kleinsten positiven $|r|$-Wert berechnet. Für $N = 4$ ergab sich $|r| = 1.094764425\ldots\cdot 10^{-47} \approx 4\cdot r_0$. Wegen der zu hohen Dichte der Gitterpunkte im *double*-Format kann das Programm aus Laufzeitgründen nicht auf den ganzen Bereich $\frac{1}{2}\sqrt{2} < \widetilde{x} < 1$ angewandt werden.

2. In der Voraussetzung 1. ist die Bedingung $\widetilde{x} > \frac{1}{2}\sqrt{2}$ zunächst willkürlich. Wir zeigen jetzt, dass für $0 < \widetilde{x} < \frac{1}{2}\sqrt{2}$ alle r negativ und kleiner $r_1 := -1.77302\ldots10^{-16}$ sind. Wegen $\widetilde{x}^2 < \frac{1}{2}$ und $0 \leq \widetilde{y} \leq \widetilde{x}$ folgt $r = \widetilde{x}^2+\widetilde{y}^2-1 < 0$, und r wird maximal für $\widetilde{y} = \widetilde{x}$, d.h. $r = 2\,\widetilde{x}^2-1 < 0$. Ist nun $\widetilde{x}_0 \in S(2,53)$ die größte Maschinenzahl, mit $\widetilde{x}_0 < \frac{1}{2}\sqrt{2}$, so folgt für die Oberschranke r_1 aller negativen r-Werte: $\quad r \leq r_1 := 2\,\widetilde{x}_0^2-1 = -1.77302\ldots10^{-16}$, d.h. im Bereich $0 < \widetilde{x} < \frac{1}{2}\sqrt{2}$ gilt: $\quad |r| \geq 1.77302\ldots10^{-16}$.

3. Im Falle $\widetilde{x} \neq 1$ und $r \neq 0$ liegen damit alle r im normalisierten Zahlenbereich!

[7] Beachten Sie, dass wegen $\widetilde{x} > \frac{1}{2}\sqrt{2}$ direkt $\widetilde{x}^2 > \frac{1}{2}$ folgt und dass auch $\widetilde{x}^2$ maximal 106 Stellen benötigt.

Nach (8.28) und wegen $|r| \geq 2^{-158}$ ist die Fehlerabschätzung durchzuführen in den beiden Bereichen

$$-0.17187501 \leq r \leq -2^{-158} \qquad \text{und} \qquad 2^{-158} \leq r \leq +0.17187501$$

Da wegen $(\widetilde{x} \neq 1 \vee \widetilde{y} \geq y_0)$ alle Werte $r = \widetilde{x}^2 + \widetilde{y}^2 - 1$ im normalisierten Bereich liegen, gilt nach (8.29) von Seite 273 für den relativen Auslesefehler ε_r die Abschätzung: $|\varepsilon_r| \leq \varepsilon(r) = 2^{-53}$. Für den relativen Fehler

$$\varepsilon_{f_4} := \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - 0.5 \odot \texttt{lnp1(r)}}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})}$$

liefert das Programm

```
// Program lnx2y2_ser.cpp for calculating the relative error by
// evaluating the term ln(1+[x^2+y^2-1]);

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval A;
    real eps;
    rel_lnp1_rel(Xi,delx,A,eps);
    return eps;
}

int main()
{
    interval X = interval(-0.17187502,comp(-0.5,-157));
    real bnd, diam = 1e-5, epsr=comp(0.5,-52);
    // epsr: Upper bound of the relative error by reading the accu.
    Max_bnd_Xi(T_x,X,epsr,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

die relative Fehlerschranke

$$|\varepsilon_{f_4}| \le \varepsilon(f_4) = 3.730016 \cdot 10^{-16}, \text{ falls } -0.17187501 \le r \le -2^{-158};$$

Im Bereich $2^{-158} \le r \le +0.17187501$ liefert das Programm `lnx2y2_ser.cpp` mit der Zeile `interval X = interval(comp(0.5,-157),0.17187502);` das Ergebnis: $|\varepsilon_{f_4}| \le \varepsilon(f_4) = 3.618424 \cdot 10^{-16}$.

Hinweise:

1. `lnp1(r)` ist das Maschinenergebnis bei Auswertung von $\ln(1+[\widetilde{x}^2+\widetilde{y}^2-1])$, wobei der exakte Wert `dot` $= \widetilde{x}^2+\widetilde{y}^2-1$ mit `r = rnd(dot)` in die *double*-Variable `r` mit der relativen Fehlerschranke $\varepsilon(r) = 2^{-53}$ ausgelesen wird.

2. Da die von Null verschiedenen Maschinenwerte `lnp1(r)` alle im normalisierten Bereich liegen, wird die Maschinenmultiplikation $0.5 \odot$ `lnp1(r)` rundungsfehlerfrei durchgeführt, so dass in der Funktion `T_x(...)` diese Multiplikation nicht zu berücksichtigen ist!

3. Beim Aufruf `Max_bnd_Xi(T_x,X,epsr,diam,bnd)` ist `epsr` $= 2^{-53}$ die Schranke des relativen Fehlers beim Auslesen des exakten Akkumulatorwertes $r = \widetilde{x}^2+\widetilde{y}^2-1$ in die `real`-Variable `r`.

Zusammenfassung:

Mit $r := \widetilde{x}^2+\widetilde{y}^2-1$ gilt im Bereich $-0.17187501 \le r \le +0.17187501$ unter der Bedingung $(\widetilde{x} \ne 1 \ \vee \ \widetilde{y} \ge y_0)$ die Abschätzung:

$$\left|\frac{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2}) - 0.5 \odot \texttt{lnp1(r)}}{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2})}\right| \le \varepsilon(f, A_4) = 3.730016 \cdot 10^{-16} \tag{8.31}$$

Im Programm `lnx2y2.cpp` auf Seite 288 wird in den Bereichen $\widetilde{s} \in A_3 = [\frac{1}{4}, 1-\frac{11}{64}]$ und $\widetilde{s} \in A_5 = [1+\frac{11}{64}, \frac{7}{4}]$ die Auswertung von $\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2})$ realisiert durch:

$$\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2}) = \frac{1}{2} \cdot \left[\ln(r) + \ln(1+\frac{r}{r_1})\right], \quad \widetilde{x}^2+\widetilde{y}^2 = r+r_1, \tag{8.32}$$

dabei wird $r+r_1$ auf dem Rechner approximiert durch den folgenden Algorithmus:

1. Exakte Addition von $\widetilde{x}^2, \widetilde{y}^2$ im Akkumulator: `dot` $= \widetilde{x}^2+\widetilde{y}^2$;
2. $r =$ `rnd(dot);` Rundung zur nächsten Rasterzahl
3. `dot` $=$ `dot` $- r$; d.h. `dot` $= \widetilde{x}^2+\widetilde{y}^2-r =: r_1$, d.h. $\widetilde{x}^2+\widetilde{y}^2 = r+r_1$;
4. `r1` $=$ `rnd(dot);` d.h. `r1` $\in S(2,53)$;

Es gilt damit $\widetilde{x}^2 + \widetilde{y}^2 = r + r_1 \approx r + \texttt{r1}$ in hoher Genauigkeit, wobei $r \in S(2,53)$ als exakt aufzufassen ist und $\texttt{r1} \approx r_1$ der fehlerbehaftete Summand ist.

Fehlerabschätzung in $A_3 = [\frac{1}{4}, 1 - \frac{11}{64}]$

Zunächst gilt wieder $\widetilde{s} \in A_3$, wobei die *double*-Zahl $\texttt{s} = \widetilde{s}$ die exakte Summe $\widetilde{x}^2 + \widetilde{y}^2$ approximiert. Wegen $\widetilde{s} = \texttt{s} = \texttt{rnd(dot)}$, mit $\texttt{dot} = s := \widetilde{x}^2 + \widetilde{y}^2$ ergibt sich[8]:

$$\texttt{s} = (\widetilde{x}^2 + \widetilde{y}^2) \cdot (1 + \varepsilon_s), \quad |\varepsilon_s| \le \varepsilon(s) = 2^{-53}; \tag{8.33}$$

Für die Fehlerabschätzungen benötigen wir nach Seite 39 stets Einschließungen der exakten Operanden, hier also eine Einschließung der s-Werte. Nach (8.33) gilt:

$$\texttt{s} = s \cdot (1 + \varepsilon_s) \ge \frac{1}{4} \quad \rightsquigarrow \quad s \ge \frac{0.25}{1 + \varepsilon_s} \ge \frac{0.25}{1 + \varepsilon(s)} > 0.24999999$$

$$\texttt{s} = s \cdot (1 + \varepsilon_s) \le 1 - \frac{11}{64} \quad \rightsquigarrow \quad s \le \frac{1 - \frac{11}{64}}{1 + \varepsilon_s} \le \frac{1 - \frac{11}{64}}{1 - \varepsilon(s)} < 0.82812501$$

Für $\widetilde{s} = \texttt{s} \in A_3$ werden die exakten Summen $s = \widetilde{x}^2 + \widetilde{y}^2$ daher eingeschlossen durch:

$$s = \widetilde{x}^2 + \widetilde{y}^2 \in S_3 := [0.24999999, 0.82812501]$$

Für die Fehlerabschätzung betrachten wir ein Teilintervall $T := [\texttt{t1}, \texttt{t2}] \subset S_3$, d.h. $\widetilde{x}^2 + \widetilde{y}^2 \equiv r + r_1 \in T$ und benötigen nach (8.32) Einschließungen der exakten Werte r und r_1. Für alle $s \in T$ sei $\widetilde{x}^2 + \widetilde{y}^2 = \texttt{t1}$ der kleinstmögliche Wert. Daraus folgt:

$$\begin{aligned} r &= \texttt{rnd(t1)}, \quad \text{d.h. } r = \texttt{t1} \cdot (1 + \varepsilon_s), \quad |\varepsilon_s| \le \varepsilon(s) = 2^{-53} \\ \texttt{r1} &= \texttt{rnd(t1} - r\texttt{)} = \texttt{rnd(} - \texttt{t1} \cdot \varepsilon_s\texttt{)}, \quad r_1 := \texttt{t1} - r = -\texttt{t1} \cdot \varepsilon_s \\ &\rightsquigarrow \quad r \ge \texttt{t1} \cdot (1 - \varepsilon(s)); \qquad r_1 \ge -\texttt{t1} \cdot \varepsilon(s); \end{aligned}$$

Für alle $s \in T$ sei $\widetilde{x}^2 + \widetilde{y}^2 = \texttt{t2}$ der größtmögliche Wert. Daraus ergibt sich:

$$\begin{aligned} r &= \texttt{rnd(t2)}, \quad \text{d.h. } r = \texttt{t2} \cdot (1 + \varepsilon_s), \quad |\varepsilon_s| \le \varepsilon(s) = 2^{-53} \\ \texttt{r1} &= \texttt{rnd(t2} - r\texttt{)} = \texttt{rnd(} - \texttt{t2} \cdot \varepsilon_s\texttt{)}, \quad r_1 := \texttt{t2} - r = -\texttt{t2} \cdot \varepsilon_s \\ &\rightsquigarrow \quad r \le \texttt{t2} \cdot (1 + \varepsilon(s)); \qquad r_1 \le \texttt{t2} \cdot \varepsilon(s); \end{aligned}$$

Wir erhalten damit für r und r_1 die folgenden Einschließungen:

$$\texttt{t1} \cdot (1 - 2^{-53}) \le r \le \texttt{t2} \cdot (1 + 2^{-53}) \tag{8.34}$$

$$-\texttt{t1} \cdot 2^{-53} \le r_1 \le \texttt{t2} \cdot 2^{-53} \tag{8.35}$$

Wir betrachten r als rundungsfehlerfrei und benötigen für die folgende Fehlerabschätzung jetzt noch eine Oberschranke des absoluten Fehlers $|\texttt{r1} - r_1|$. Nimmt man zunächst an, dass die $\texttt{r1} \in S(2,53)$ alle im normalisierten Bereich liegen, so gilt wegen $\texttt{t2} \in [0.24999999, 0.82812501]$ nach Definition des relativen Fehlers:

[8] Die Summen $\widetilde{x}^2 + \widetilde{y}^2$ liegen jetzt alle im normalisierten Bereich, und daher gilt $\varepsilon(s) = 2^{-53}$.

$$|\mathtt{r1} - r_1| = |\varepsilon_s| \cdot |r_1| \leq \varepsilon(s) \cdot |r_1| \leq \varepsilon(s) \cdot (\mathtt{t2} \cdot 2^{-53}) = \mathtt{t2} \cdot 2^{-106}$$

Falls die `r1` jedoch in den denormalisierten Bereich fallen, so ist der absolute Fehler beschränkt durch $\mathtt{MinReal} = 2^{-1022} < \mathtt{t2} \cdot 2^{-106}$, so dass der obige Wert $\mathtt{t2} \cdot 2^{-106}$ auch dann eine gültige Oberschranke bleibt! Der bei Auswertung von (8.32) auftretende relative Fehler

$$\varepsilon_{f_3} := \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - 0.5 \odot [\mathtt{ln}(r) \oplus \mathtt{lnp1}(\mathtt{r1} \oslash r)]}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})}$$

wird abgeschätzt mit dem **C-XSC** Programm `lnx2y2_A3.cpp`

```
// Program lnx2y2_A3.cpp for calculating the relative error by
// evaluating the term:  0.5*[ln(r) + ln(1+(r1/r)];  x^2+y^2 = r+r1;
// x^2+y^2 calculated in the accumulator dot = x^2+y^2;
// r = rnd(dot);  dot = dot-r;  r1 = rnd(dot);

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout

using namespace cxsc;
using namespace std;

const real epsS   = comp(0.5,-52);  // epsS   = 2^(-53);
const real one_eS = subd(1,epsS);   // one_eS = 1-epsS;
const real onepeS = addu(1,epsS);   // onepeS = 1+epsS;

real T_x(const interval& Xi, const real& delx)
{
    interval R,R1,A,B,Q;
    real absA,absB,absQ,eps,x,y,abs_r1,IXi=Inf(Xi),SXi=Sup(Xi);
    x = muld(IXi,one_eS);
    y = mulu(SXi,onepeS);
    R = interval(x,y);   // R: inclusion of the r values
    x = muld(-IXi,epsS);
    y = mulu(SXi,epsS);
    R1 = interval(x,y);  // R1: inclusion of the r1 values

    abs_ln_abs(R,0,A,absA);
```

```
    abs_r1=SXi;  times2pown(abs_r1,-106);
    // abs_r1: absolute error of the r1
    abs_divh1(R1,abs_r1,R,0,Q,absQ);
    abs_lnp1_abs(Q,absQ,B,absB);

    rel_addh1(A,absA,B,absB,R,eps);
    return eps;
}

int main()
{
    interval X;  string("0.24999999,0.82812501") >> X;
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Das obige Programm liefert für alle $s = \widetilde{x}^2 + \widetilde{y}^2 \in S_3 = [0.24999999, 0.82812501] \supset A_3$ mit $\widetilde{f_3} := 0.5 \odot [\texttt{ln}(r) \oplus \texttt{lnp1}(\texttt{r1} \oslash r)]$ für den relativen Auswertefehler die folgende Abschätzung:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f_3}}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, A_3) = 5.160435 \cdot 10^{-16} \tag{8.36}$$

Anmerkungen:

1. Mit der obigen Programmzeile

   ```
   interval X;  string("0.24999999,0.82812501") >> X;
   ```

   erreicht man dass $S_3 = [0.24999999, 0.82812501]$ durch das Maschinenintervall `X` garantiert eingeschlossen wird: $S_3 \subset$ `X`.

2. Da die Funktionswerte $\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})$ für $s \in S_3$ alle im normalisierten Bereich liegen, wird bei Auswertung von $f_3 := \frac{1}{2} \cdot [\ln(r) + \ln(1 + \frac{r_1}{r})]$ die Multiplikation mit $\frac{1}{2}$ rundungsfehlerfrei durchgeführt. Im obigen Programm wird daher in der Funktion `T_x(...)` diese Multiplikation nicht berücksichtigt!

3. Die direkte Auswertung von $\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) = \frac{1}{2} \cdot \ln(\widetilde{x}^2 + \widetilde{y}^2)$ liefert für $s \in S_3$ mit $8.8268 \cdot 10^{-16}$ eine deutlich größere Fehlerschranke. Bei der zeitaufwendigeren

Auswertung von f_3 erhalten wir jedoch eine deutlich kleinere Fehlerschranke, da $\ln(r)$ mit dem rundungsfehlerfreien Argument $r \in S(2,53)$ ausgewertet wird.

Fehlerabschätzung in $A_5 = [1 + \frac{11}{64}, \frac{7}{4},]$

Die Fehlerabschätzung in diesem Teilbereich erfolgt völlig analog zum Bereich A_3. Lediglich die exakten Summen $s := \widetilde{x}^2 + \widetilde{y}^2$ werden jetzt eingeschlossen durch:

$$s = \widetilde{x}^2 + \widetilde{y}^2 \in S_5 := [1.17187499999, 1.7500001] \supset A_5$$

Die Abschätzung des relativen Auswertefehlers erfolgt wieder mit dem gleichen Programm von Seite 279, in dem die Zeile `interval X; string("0.24999999,...` zu ersetzen ist durch: `interval X; string("1.17187499999,1.7500001") >> X;` Das Programm liefert dann für $s = \widetilde{x}^2 + \widetilde{y}^2 \in S_5 = [1.17187499999, 1.7500001] \supset A_5$ mit $\widetilde{f_5} := 0.5 \odot [\texttt{ln}(r) \oplus \texttt{lnp1}(\texttt{r1} \oslash r)]$ für den relativen Auswertefehler die folgende Abschätzung:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f_5}}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, A_5) = 5.160563 \cdot 10^{-16} \tag{8.37}$$

Im Programm `lnx2y2` von Seite 288 bleiben jetzt noch die beiden folgenden Bereiche $2^{-39} \leq \widetilde{x}^2 + \widetilde{y}^2 \ \wedge \ \widetilde{s} \leq \frac{1}{4}$ und $\frac{7}{4} \leq \widetilde{s} \ \wedge \ \widetilde{x}^2 + \widetilde{y}^2 \leq 2^{40}$, in denen $\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})$ wie folgt ausgewertet wird:

$$\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) = \frac{1}{2} \cdot \ln(\widetilde{x}^2 + \widetilde{y}^2) \approx 0.5 \odot \texttt{ln}(\texttt{s})$$

Die Summe $s := \widetilde{x}^2 + \widetilde{y}^2 =$ `dot` wird im Akkumulator zunächst rundungsfehlerfrei berechnet. Danach wird diese Summe mit $\widetilde{s} =$ `s` $=$ `rnd(dot)` zur nächsten *double*-Zahl $\widetilde{s} =$ `s` gerundet, und die **C-XSC** Funktion `ln(...)` wird dann mit diesem Argument `s` ausgewertet. Da alle s im normalisierten Bereich liegen, gilt $\texttt{s} = s \cdot (1 + \varepsilon_s)$, mit $|\varepsilon_s| \leq \varepsilon(s) = 2^{-53}$, und die exakten Summen $s := \widetilde{x}^2 + \widetilde{y}^2$ werden in den beiden Bereichen eingeschlossen durch:

$$s \in S_2 := [2^{-39}, 0.25000001] \supset A_2, \qquad s \in S_6 := [1.74999999, 2^{40}] \supset A_6; \tag{8.38}$$

Fehlerabschätzung in $A_2 = [2^{-39}, \frac{1}{4}]$

Die exakten Summen $s := \widetilde{x}^2 + \widetilde{y}^2$ werden nach (8.38) in diesem Bereich eingeschlossen durch $s \in S_2 := [2^{-39}, 0.25000001] \supset A_2$, und der relative Fehler

$$\varepsilon_{f_2} := \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - 0.5 \odot \texttt{ln}(\texttt{s})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})}, \qquad \texttt{s} = \texttt{rnd}(\widetilde{x}^2 + \widetilde{y}^2)$$

wird abgeschätzt mit dem **C-XSC** Programm `lnx2y2_A2.cpp`

```
// Program lnx2y2_A2.cpp for calculating the relative error by
// evaluating the term ln(x^2+y^2); x^2+y^2 is exactly
// calculated in the accumulator, which can be read out
// with the relative error bound epsS=2^(-53);

#include "abs_relh.hpp" // For abs_addh1, ...
#include "bnd_util.hpp" // For Max_bnd_Xi()
#include <iostream>     // For cout
using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval A;
    real eps;
    rel_ln_rel(Xi,delx,A,eps);
    return eps;
}
int main()
{
    interval X; string("[1.8189894E-12,0.25000001]") >> X;
    real bnd, diam = 1e-5,
    delx = comp(0.5,-52); // delx = 2^(-53);
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative error bound = " << bnd << endl;
}
```

Das obige Programm liefert für alle $s = \widetilde{x}^2 + \widetilde{y}^2 \in S_2 = [2^{-39}, 0.25000001] \supset A_2$ mit $\widetilde{f_2} := 0.5 \odot \texttt{ln(s)}$ für den relativen Auswertefehler die folgende Abschätzung:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2}) - \widetilde{f_2}}{\ln(\sqrt{\widetilde{x}^2+\widetilde{y}^2})} \right| \leq \varepsilon(f, A_2) = 3.740657 \cdot 10^{-16} \tag{8.39}$$

Anmerkungen:

1. Die Zeile `interval X; string("[1.8189894E-12,0.25000001]") >> X;` garantiert die Einschließung $S_2 := [2^{-39}, 0.25000001] \subset$ `X`.
2. Da die Maschinenwerte `ln(s)` stets im normalisierten Zahlenbereich liegen und die Multiplikation $0.5 \odot$ `ln(s)` daher rundungsfehlerfrei erfolgt, wird diese in der Funktion `T_x(...)` auch nicht berücksichtigt.

Fehlerabschätzung in $A_6 = [\frac{7}{4}, 2^{+40}]$

Die exakten Summen $s := \widetilde{x}^2 + \widetilde{y}^2$ werden nach (8.38) in diesem Bereich eingeschlossen durch $s \in S_6 := [1.74999999, 2^{40}] \supset A_6$, und der relative Fehler

$$\varepsilon_{f_6} := \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - 0.5 \odot \texttt{ln(s)}}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})}, \qquad \texttt{s} = \texttt{rnd}(\widetilde{x}^2 + \widetilde{y}^2)$$

wird abgeschätzt mit dem **C-XSC** Programm `lnx2y2_A2.cpp` von Seite 283, in dem lediglich die Zeile `interval X; string("[1.8189894E-12,...` zu ersetzen ist durch: `interval X; string("[1.74999999,1.09951163E+12]") >> X;`

Das Programm liefert dann für alle $s = \widetilde{x}^2 + \widetilde{y}^2 \in S_6 = [1.74999999, 2^{40}] \supset A_6$ mit $\widetilde{f_6} := 0.5 \odot \texttt{ln(s)}$ für den relativen Auswertefehler die folgende Abschätzung:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f_6}}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f, A_6) = 4.923703 \cdot 10^{-16} \tag{8.40}$$

Damit haben wir in allen Teilbereichen A_i eine absolute oder relative Fehlerschranke bei Auswertung der jeweiligen Funktionen $f_i(\widetilde{x}, \widetilde{y})$ berechnet. Ich habe dabei die notwendigen Schritte und alle verwendeten Hilfsprogramme möglichst ausführlich beschrieben in der Hoffnung, dass der Leser nach einem sorgfältigem Studium dieses Beispiels in der Lage sein wird, eine Fehlerabschätzung auch für eigene Algorithmen selbständig und erfolgreich durchführen zu können.

Zusammenfassung:

Im Falle $\widetilde{x} = 1$ und $0 \leq \widetilde{y} \leq y_0$ gilt mit $f_{4a}(y) := \frac{1}{2} \cdot y^2$

$$\left| \ln(\sqrt{1 + \widetilde{y}^2}) - \widetilde{f}_{4a}(\widetilde{y}) \right| \leq \Delta(f_4) = 2.225075 \cdot 10^{-308} \tag{8.41}$$

Sonst gilt für $|\widetilde{x}| + |\widetilde{y}| > 0$ in allen Teilbereichen A_i die Abschätzung:

$$\left| \frac{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2}) - \widetilde{f}_i(\widetilde{x}, \widetilde{y})}{\ln(\sqrt{\widetilde{x}^2 + \widetilde{y}^2})} \right| \leq \varepsilon(f) = 5.160563 \cdot 10^{-16} \tag{8.42}$$

$y_0 \in S(2, 53)$ ist auf Seite 270 als Quotient zweier Maschinenzahlen angegeben, wobei diese Maschinenzahlen in dezimaler Form dargestellt werden können. y_0 wird nur im Quellcode der Intervallversion benötigt, vergleiche Seite 288.

8.2.2 Intervallargumente

Durch die Vereinbarung `interval x,y;` werden Maschinenintervalle $\mathtt{x},\mathtt{y} \in IR$ vorgegeben, und für alle reellen $x \in \mathtt{x}$ und $y \in \mathtt{y}$ ist eine Maschineneinschließung `W` des Wertebereichs

$$W_f := \left\{ f(x,y) \in I\!R \;\middle|\; f(x,y) := \ln(\sqrt{x^2+y^2}) \;\wedge\; x \in \mathtt{x} \;\wedge\; y \in \mathtt{y} \right\} \subset \mathtt{W} \tag{8.43}$$

gesucht. Wegen der Quadrate x^2, y^2 kann man sich durch[9] `x=abs(x)` und `y=abs(y)` auf die nicht negativen Argumente $x \in \mathtt{x}$ und $y \in \mathtt{y}$ beschränken. Da die Funktion $\ln(\sqrt{s})$ in Abhängigkeit von $s = x^2 + y^2$ streng monoton wächst, ist die Implementierung einer Einschließung $\mathtt{W} \in IR$ relativ einfach. Zur Berechnung von `Sup(W)` ruft man die **C-XSC** Funktion `real ln_sqrtx2y2(const real& x, const real& y)` mit den Argumenten `Sx=Sup(x)` und `Sy=Sup(y)` auf und erhält so mit der Anweisung `f = ln_sqrtx2y2(Sx,Sy)` den Maschinenwert $\mathtt{f} \in S(2,53)$ als Approximation des exakten Funktionswertes $\ln(\sqrt{\mathtt{Sx}^2+\mathtt{Sy}^2}) \approx \mathtt{f}$. Liegt `f` im normalisierten Bereich, so muss im Falle $\mathtt{f} \geq 0$ dieser Wert noch multipliziert werden mit $\mathtt{q\_lnx2y2p} \in S(2,53)$, wobei `q_lnx2y2p = 4503599627370502.0 / 4503599627370496.0;` mit Hilfe der Funktion `eps2fractions(str,Z1p,N1p,Z1m,N1m)` berechnet wird. Der string `str = "5.160563E-16"` ist dabei die relative Fehlerschranke von Seite 284 und die Rückgabewerte `Z1p=4503599627370502.0, N1p=4503599627370496.0` sind Zähler und Nenner von `q_lnx2y2p`. Wir erhalten so mit `u2 = f*q_lnx2y2p = Sup(W)` die gesuchte Oberschranke `Sup(W)` der Einschließung $\mathtt{W} \supset W_f$. Weitere Einzelheiten zur Funktion `eps2fractions(...)` findet man auf Seite 81. Liegt `f` jedoch im denormalisierten Bereich, so muss zur Berechnung von `Sup(W)` noch die absolute Fehlerschranke `ln_x2y2_abs = 2.225076E-308` addiert werden. Im Vergleich zur Schranke $\Delta(f_4) = 2.225075 \cdot 10^{-308}$ von Seite 284 wurde dieser Wert so aufgerundet, dass bei der Addition `u2 = f + ln_x2y2_abs;` keine zeitaufwendige gerichtete Rundung notwendig ist, d.h. auch wenn der Rechner im Abrundungsmodus laufen sollte, wird durch die genannte Addition mit `u2` stets eine garantierte Oberschranke ermittelt.

Die Berechnung einer Unterschranke `Inf(W)` erfolgt ganz analog. Wenn jedoch im Falle `(Ix==1 && Iy<b0 || Iy==1 && Ix<b0)` die mit `f = ln_sqrtx2y2(Ix,Iy)` berechneten Funktionswerte `f` im denormalisierten Bereich liegen und negativ sind, so kann die Unterschranke `u1 = f - ln_x2y2_abs` auf Null gesetzt werden, denn für die exakten Funktionswerte gilt dann z.B. $f(x,y) = f(1,y) = \frac{1}{2}\ln(1+y^2) \geq 0$. Weitere Einzelheiten dazu findet man im Quelltext ab Seite 288. Die Funktion zur Berechnung des einschließenden Intervalls `W` wird deklariert durch:

```
interval ln_sqrtx2y2(const interval& x, const interval& y) throw();
```

[9] Mit `x1=abs(x)` erhält man das Intervall `x1` der Beträge aller $x \in \mathtt{x}$.

8.2.3 Numerische Ergebnisse

In diesem Abschnitt werden zu gegebenen Intervallargumenten `x,y` Einschließungen $\mathtt{W} \supset W_f$ berechnet, wobei W_f in (8.43) definiert ist. Mit Hilfe der Funktion

```
interval ln_sqrtx2y2(const interval& x, const interval& y) throw();
```

liefert der Aufruf `W = ln_sqrtx2y2(x,y);` die gesuchte Einschließung $\mathtt{W} \in IR$, vergleichen Sie dazu den Quelltext ab Seite 288. Beachten Sie, dass die berechneten binären Intervalle $\mathtt{W} \in IR$ durch die mit `cout << ln_sqrtx2y2(x,y) << endl;` ausgegebenen dezimalen Intervalle stets eingeschlossen werden.

1. $\mathtt{x} = [-1, 1]$ und $\mathtt{y} = [-1, 1]$ liefern die Fehlermeldung

   ```
   real ln_sqrtx2y2(const real&, const real&): STD_FKT_OUT_OF_DEF
   ```

   da die Bedingung $|x| + |y| > 0$ nicht erfüllt ist.

2. $\mathtt{x} = [-1, -1],\ \mathtt{y} = [0, 0]$
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [0.000000000000000 \cdot 10^0, 0.000000000000000 \cdot 10^0]$

3. $\mathtt{x} = [1, 1],\ \mathtt{y} = [1, 1]$
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [3.465735902799723 \cdot 10^{-1}, 3.465735902799731 \cdot 10^{-1}]$

4. $\mathtt{x} = [2, 2],\ \mathtt{y} = [2, 2]$
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [1.039720770839916 \cdot 10^0, 1.039720770839920 \cdot 10^0]$

5. $\mathtt{x} = [1, 2],\ \mathtt{y} = [1, 2]$
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [3.465735902799723 \cdot 10^{-1}, 1.039720770839920 \cdot 10^0]$

6. $\mathtt{x} = [0.8, 0.8],\ \mathtt{y} = [0.6, 0.6]$
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [-6.661338147750948 \cdot 10^{-17}, 8.881784197001264 \cdot 10^{-17}]$
   Beachten Sie bitte, dass jetzt `x` und `y` keine Punktintervalle sind und 0.8 bzw. 0.6 optimal einschließen. Wegen $0.8^2 + 0.6^2 = 1$ muss daher `W` die Zahl Null zwar einschließen, es darf aber nicht gelten $\mathtt{W} = [0, 0]$.

7. ```
   x = interval(9007199254740988.0 / 9007199254740992.0);
   y = interval(9007199254740991.0 / 3022314549036572936765440.0);
   ```
   $\rightsquigarrow \quad W_f \subset \mathtt{W} \subset [5.473822126268811 \cdot 10^{-48}, 5.473822126268824 \cdot 10^{-48}]$
   Vergleichen Sie bitte die Anmerkung 1. auf Seite 275. Für die Punktintervalle `x,y` liefert Maple das Ergebnis:

   $$\ln(\sqrt{x^2 + y^2}) = 5.473822126268816683295818684726234\ldots \cdot 10^{-48}$$
   ```

8. $\texttt{x} = [10^{100}, 10^{100}],\ \texttt{y} = [10^{100}, 10^{100}]$
   $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [2.306050828896843 \cdot 10^2, 2.306050828896849 \cdot 10^2]$
   Beachten Sie bitte, dass `x,y` keine Punktintervalle sind, die jedoch den Wert 10^{100} optimal einschließen, d.h. `Inf(x)` und `Sup(x)` aus $S(2,53)$ sind die zu 10^{100} nächstgelegenen Rasterzahlen.

9. $\texttt{x} = [10^{308}, 10^{308}],\ \texttt{y} = [10^{308}, 10^{308}]$
   $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [7.095427822324453 \cdot 10^2, 7.095427822324470 \cdot 10^2]$
   Auch jetzt sind `x,y` keine Punktintervalle; beachten Sie, dass die Berechnung der Quadratsumme $s = x^2 + y^2$ hier nicht zum Overflow führt!

10. $\texttt{x} = [10^{-100}, 10^{-100}],\ \texttt{y} = [10^{-100}, 10^{-100}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [-2.299119357091250 \cdot 10^2, -2.299119357091244 \cdot 10^2]$
    Beachten Sie bitte, dass `x,y` keine Punktintervalle sind, die jedoch den Wert 10^{-100} optimal einschließen.

11. $\texttt{x} = [10^{-308}, 10^{-308}],\ \texttt{y} = [10^{-308}, 10^{-308}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [-7.088496350518871 \cdot 10^2, -7.088496350518854 \cdot 10^2]$
    Auch jetzt sind `x,y` keine Punktintervalle; beachten Sie, dass die Berechnung der Quadratsumme $s = x^2 + y^2$ hier nicht zum Underflow führt!

12. $\texttt{x} = [2^{-1022}, 2^{-1022}],\ \texttt{y} = [2^{-1022}, 2^{-1022}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [-7.080498449419851 \cdot 10^2, -7.080498449419834 \cdot 10^2]$
    `x,y` sind jetzt Punktintervalle, die jeweils die kleinste positive, normalisierte Rasterzahl `MinReal` $:= 2^{-1022} \in S(2,53)$ einschließen.

13. $\texttt{x} = [2^{-1074}, 2^{-1074}],\ \texttt{y} = [2^{-1074}, 2^{-1074}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [-7.440934983311023 \cdot 10^2, -7.440934983311005 \cdot 10^2]$
    `x,y` sind jetzt jedoch wieder Punktintervalle, die jeweils die kleinste positive Rasterzahl `minreal` $:= 2^{-1074} \in S(2,53)$ einschließen.

14. $\texttt{x} = [1,1],\ \texttt{y} = [2^{-1022}, 2^{-1022}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [0.000000000000000 \cdot 10^0, 2.225076000000001 \cdot 10^{-308}]$
    Beachten Sie, dass der Funktionswert $\ln(\sqrt{x^2+y^2}) = \frac{1}{2}\ln(1+2^{-2044})$ jetzt im Unterlaufbereich liegt und daher wegen der absoluten Fehlerschranke $\Delta(f_4) = 2.225075 \cdot 10^{-308}$ von Seite 284 nur grob eingeschlossen werden kann.

15. $\texttt{x} = [2^{-1022}, 2^{-1022}],\ \texttt{y} = [1,1],$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [0.000000000000000 \cdot 10^0, 2.225076000000001 \cdot 10^{-308}]$
    Beachten Sie, dass ein Vertauschen der Argumente `x,y` die Einschließung der Funktionswerte nicht verändern darf.

16. $\texttt{x} = [1,1],\ \texttt{y} = [0, 2^{-500}]$
    $\rightsquigarrow \quad W_f \subset \texttt{W} \subset [0.000000000000000 \cdot 10^0, 4.666318092516101 \cdot 10^{-302}]$

8.2.4 Quelltext für Punkt- und Intervallargumente

Der nachfolgende Quelltext enthält die vollständige Implementierung der Funktionen mit Punkt- und Intervall-Argumenten:

```
// Program:  lnx2y2_source.cpp  -----------------------------------
#include <iostream>
#include <rmath.hpp>
#include <interval.hpp>

using namespace cxsc;
using namespace std;

// With the following b0
real b0 = 6369051672525773.0 / 30191699398572330817932436647906151127
          33536976333152342700965040196499329913719081668901380142127
          01403317470002461107591981646770393983410604914740114615683
          49195162615808.0;
// it holds:
// 1. b < bo  ==> g(b) := (0.5*b)*b < MinReal with rounding downwards
// 2. b >= b0 ==> g(b) := (0.5*b)*b >=MinReal with arbitrary rounding
//                                    modus by the two multiplications.

dotprecision dot;
// Error bounds for the interval function:
real ln_x2y2_abs(2.225076E-308); // Absolute error bond
real q_lnx2y2p(4503599627370502.0 / 4503599627370496.0); // 1+e(f)
real q_lnx2y2m(9007199254740984.0 / 9007199254740992.0); // 1-e(f)

real ln2_1067(6505485212531678.0 / 8796093022208.0); // 1067*ln(2)
// Exponents of the interval bounds:
int B_lnx2y2_1[22] = {21, 31, 51, 101, 151, 201, 251, 301, 351, 401,
                      451, 501, 551, 601, 651, 701, 751, 801, 851,
                      901, 951, 1025};
int B_lnx2y2_2[22] = {-1021,-949,-899,-849,-799,-749,-699,-649,-599,
                      -549,-499,-449,-399,-349,-299,-249,-199,-149,
                      -99,-49,-29,-19};
// Optimal values N for N*ln(2):
int B_lnx2y2_N1[21] ={20, 40, 61, 122, 160, 229, 259, 320, 366, 427,
                      488, 518, 549, 610, 671, 732, 763, 825, 885,
                      945, 976};
```

```
real B_lnx2y2_c1[21] =
{
    7804143460206699.0 / 562949953421312.0, // N*ln(2) with N = 20;
    7804143460206699.0 / 281474976710656.0, // N*ln(2) with N = 40;
    5950659388407608.0 / 140737488355328.0, // N*ln(2) with N = 61;
    5950659388407608.0 / 70368744177664.0,  // N*ln(2) with N = 122;
    7804143460206699.0 / 70368744177664.0,  // N*ln(2) with N = 160;
    5584840163710419.0 / 35184372088832.0,  // N*ln(2) with N = 229;
    6316478613104797.0 / 35184372088832.0,  // N*ln(2) with N = 259;
    7804143460206699.0 / 35184372088832.0,  // N*ln(2) with N = 320;
    8925989082611412.0 / 35184372088832.0,  // N*ln(2) with N = 366;
    5206826964856657.0 / 17592186044416.0,  // N*ln(2) with N = 427;
    5950659388407608.0 / 17592186044416.0,  // N*ln(2) with N = 488;
    6316478613104797.0 / 17592186044416.0,  // N*ln(2) with N = 518;
    6694491811958559.0 / 17592186044416.0,  // N*ln(2) with N = 549;
    7438324235509510.0 / 17592186044416.0,  // N*ln(2) with N = 610;
    8182156659060461.0 / 17592186044416.0,  // N*ln(2) with N = 671;
    8925989082611412.0 / 17592186044416.0,  // N*ln(2) with N = 732;
    4652001140732587.0 / 8796093022208.0,   // N*ln(2) with N = 763;
    5030014339586349.0 / 8796093022208.0,   // N*ln(2) with N = 825;
    5395833564283538.0 / 8796093022208.0,   // N*ln(2) with N = 885;
    5761652788980727.0 / 8796093022208.0,   // N*ln(2) with N = 945;
    5950659388407608.0 / 8796093022208.0    // N*ln(2) with N = 976;
};

int Interval_Nr(int* v, const int& n, const int& ex)
// n>0 subintervals:   |...\|...\|...\ ..... |...|
// subinterval Nr.:      0    1    2   .....  n-1
{
    int i=0,j=n,k;  // n>0:  Number of subintervals
    do {
        k = (i+j)/2;
        if (ex < v[k]) j = k-1;
        else i = k+1;
    } while(i<=j);
    return j;  // x with ex=expo(x) lies in the subinterval number j
}

real ln_sqrtx2y2(const real& x, const real& y)
```

```
                                throw(STD_FKT_OUT_OF_DEF)
// ln( sqrt(x^2+y^2) ) == 0.5*ln(x^2+y^2); Blomquist, 21.11.03;
{
    int j,N;
    real a,b,r,r1;
    a = sign(x)<0 ? -x : x;  // a = |x| >= 0;
    b = sign(y)<0 ? -y : y;  // b = |y| >= 0;
    int exa=expo(a), exb=expo(b), ex;
    if (b > a)
    {
        r = a;   a = b;   b = r;
        ex = exa;   exa = exb;   exb = ex;
    }
    // It holds now:   0 <= b <= a
    if (sign(a)==0)
        cxscthrow(STD_FKT_OUT_OF_DEF
                    ("real ln_sqrtx2y2(const real&, const real&)"));
    if (exa>20) // to avoid overflow by calculating a^2 + b^2
    {  // a>=2^(20):
        j = Interval_Nr(B_lnx2y2_1,21,exa); // j: No. of subinterval
        N = B_lnx2y2_N1[j];    // N: Optimal int value
        if (exb-exa > -25)
        {   // For (exb-exa>-25) we use the complete term:
            // N*ln(2) + [ln(2^(-N)*a)+0.5*ln(1+(b/a)^2)]
            b = b/a;  // a > 0
            b = lnp1(b*b);
            times2pown(b,-1);  // exact division by 2
            times2pown(a,-N);
            r = b + ln(a); // [ ... ] calculated!
            r += B_lnx2y2_c1[j];
        }
        else { // For (exb-exa<=-25) only two summands!:
            times2pown(a,-N);
            r = ln(a) + B_lnx2y2_c1[j];
        }
    }
    else  // exa<=20 or a<2^(20):
    {     // Now calculation of a^2+b^2 without overflow:
        if (exa<=-20) // to avoid underflow by calculating a^2+b^2
            if (exa<=-1022) // a in the denormalized range
```

```
        {
            r = b/a;
            r = lnp1(r*r);  times2pown(r,-1); // r: 0.5*ln(1+..)
            times2pown(a,1067);
            r += ln(a);     // [ .... ] ready
            r -= ln2_1067;  // rel. error = 2.459639e-16;
        }
        else // MinReal=2^(-1022) <= a < 2^(-20)
        {    // Calculating the number j of the subinterval:
            j = 20 - Interval_Nr(B_lnx2y2_2,21,exa);
            r = a;  times2pown(r,B_lnx2y2_N1[j]);
            r = ln(r);  // r: ln(2^N*a);
            if (exb-exa > -25) { // calculating the complete term
                b = b/a;
                a = lnp1(b*b);
                times2pown(a,-1);
                r += a;  // [ ... ] ready now
            }
            // We now have: exb-exa<=-25,  ==>  b/a <= 2^(-24);
            r -= B_lnx2y2_c1[j]; // 0.5*ln(1+(b/a)^2) neglected!
            // relative error = 4.524090e-16 in both cases;
        }
    else // calculation of a^2+b^2 without overflow or underflow:
    {   // exa>-20  respective  a>=2^(-20):
        dot = 0;
        accumulate(dot,a,a);
        accumulate(dot,b,b);  // dot = a^2+b^2, exact!
        real s = rnd(dot);    // s = a^2 + b^2, rounded!
        if (s>=0.25 && s<=1.75)
            if (s>=0.828125 && s<=1.171875)
            { // Series:
                if (a==1 && exb<=-28)
                {
                    r = b;  times2pown(r,-1);
                    r *= b;
                }
                else {
                    dot -= 1;
                    r = rnd(dot); // r = a^2+b^2-1 rounded!
```

```
                    r = lnp1(r);
                    times2pown(r,-1);
                }
            }
            else { // Reading dot = a^2+b^2 twice:
                r = rnd(dot);
                dot -= r;
                r1 = rnd(dot); // a^2+b^2 = r+r1, rounded!
                r1 = lnp1(r1/r);
                r = ln(r) + r1;
                times2pown(r,-1); // exact division by 2
            }
        else { // calculating straight from: 0.5*ln(x^2+y^2)
            r = ln(s);
            times2pown(r,-1);
        }
    }
}
return r;
} // ln_sqrtx2y2

interval ln_sqrtx2y2(const interval& x, const interval& y) throw()
// ln( sqrt(x^2+y^2) ) == 0.5*ln(x^2+y^2);   Blomquist, 22.11.03;
{
    interval ax=abs(x), ay=abs(y);
    real Ix=Inf(ax), Sx=Sup(ax), Iy=Inf(ay), Sy=Sup(ay),f,u1,u2;
    // Calculating the lower bound u1:
    f = ln_sqrtx2y2(Ix,Iy);
    if (Ix==1 && Iy<b0 || Iy==1 && Ix<b0) {
    // f in the denormalized range!
        u1 = f - ln_x2y2_abs;   // directed rounding not necessary!
        if (sign(u1)<0) u1 = 0;
    } else  u1 = (sign(f)<0) ? f*q_lnx2y2p : f*q_lnx2y2m;
    // Calculating the upper bound u2:
    if (Ix==Sx && Iy==Sy) // x and y are point-intervals
        if (Sx==1 && Sy<b0 || Sy==1 && Sx<b0) {
        // f in the denormalized range!
            u2 = (Sy==0 || Sx==0) ? f : f+ln_x2y2_abs;
        } else  u2 = (sign(f)<0) ? f*q_lnx2y2m : f*q_lnx2y2p;
```

```
    else // x or y is no point-interval:
    {
        f = ln_sqrtx2y2(Sx,Sy);
        if (Sx==1 && Sy<b0 || Sy==1 && Sx<b0)
        // f in the denormalized range!
            u2 = (sign(Sy)==0 || sign(Sx)==0) ? f : f+ln_x2y2_abs;
        else  u2 = (sign(f)<0) ? f*q_lnx2y2m : f*q_lnx2y2p;
    }
    return interval(u1,u2);
}
int main() {
/*  real x,y; // Function call with point arguments
    while (1) {
        cout << "real x = ? ";  cin >> x;
        cout << "real y = ? ";  cin >> y;
        cout << SetPrecision(18,18) << Scientific
             << "ln_sqrtx2y2(x,y) = " << ln_sqrtx2y2(x,y)
             << endl << endl;
    }
*/
    interval x,y; // Function call with interval arguments
    while (1) {
        cout << "interval x = ? ";  cin >> x;
        cout << "interval y = ? ";  cin >> y;

        cout << SetPrecision(16,15) << Scientific
             << "ln_sqrtx2y2(x,y) = " << ln_sqrtx2y2(x,y)
             << endl << endl;
    }
} // main
//--------------------------------------------------------------------
```

Anmerkungen:

1. Im obigen Quelltext ist der Nenner von `b0` aus drucktechnischen Gründen in vier getrennten Zeilen angegeben.

2. Mit der Funktion `int Interval_Nr(int* v,const int& n,const int& ex)` kann man zu einem Feld `v` aus `n` + 1 *integer* Zahlen, mit denen `n` halboffene Teilintervalle definiert werden und zu einem vorgegebenen Exponenten `ex` die Nummer des Intervalls `[v[j],v[j+1])` bestimmt werden, in dem `ex` liegt. Das erste Teilintervall erhält die Nummer 0 und das letzte die Nummer `n` − 1.

8.3 $\sqrt{x^2-1}$

Als weiteres Beispiel zur Fehlerabschätzung betrachten wir die reelle Funktion

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

Für alle Maschinenzahlen[10] $\widetilde{x} \geq \texttt{succ}(1.0)$ ist eine garantierte Oberschranke $\varepsilon(f)$ des relativen Fehlers

$$\left| \frac{\sqrt{\widetilde{x}^2-1} - \widetilde{f}(\widetilde{x})}{\sqrt{\widetilde{x}^2-1}} \right| \leq \varepsilon(f), \quad \widetilde{x} \in S(2,53), \quad \widetilde{x} \geq \texttt{succ}(1.0); \tag{8.44}$$

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

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

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

8.3.1 Punktargumente

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

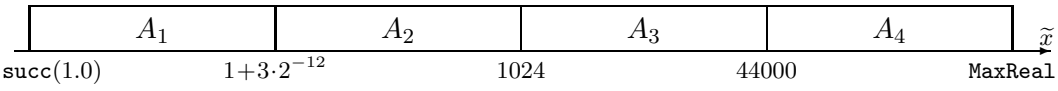


Abbildung 8.2: Teilintervalle A_i der Maschinenzahlen $\widetilde{x} \in S(2,53)$

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

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

$$y_1 := \frac{1}{2}\cdot\left(\widetilde{y}_0 + \frac{\widetilde{x}^2-1}{\widetilde{y}_0}\right) \equiv \widetilde{y}_0 + \frac{1}{2}\cdot\frac{(\widetilde{x}^2-1)-\widetilde{y}_0^2}{\widetilde{y}_0} =: g(\widetilde{x}) \tag{8.45}$$

Eine wirkliche Verbesserung der Genauigkeit erhält man jedoch nur[11], wenn man in (8.45) bei der Auswertung des rechten Terms $g(\widetilde{x})$ darauf achtet, dass der Auswertefehler des zweiten Summanden mit dem Faktor 1/2 hinreichend klein bleibt. In diesem Fall ist dann auf der Maschine eine Summe auszuwerten, deren erster Summand $\widetilde{y}_0$ als exakt aufzufassen ist und deren betragsmäßig sehr viel kleinerer zweiter Summand mit nur einem kleinen Fehler behaftet ist. Nach Seite 229 ist dann der durch

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

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

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

$$\widetilde{a} = (\widetilde{x}^2-1) + \delta_a, \quad |\delta_a| \le \Delta(a)$$

$$\widetilde{y}_0 := \texttt{sqrt}(\widetilde{a}) = \sqrt{\widetilde{x}^2-1}\cdot(1+\varepsilon_0), \quad |\varepsilon_0| \le \varepsilon(0), \tag{8.46}$$

wobei `sqrt(...)` die in **C-XSC** definierte Wurzelfunktion ist. Die Berechnung der relativen Fehlerschranke $\varepsilon(0)$ erfolgt dabei mit Hilfe der Funktion `rel_sqrt_abs(...)`, vgl. Sie dazu bitte die dritte Zeile in der letzten Spalte der Tabelle 6.1 auf Seite 153.

Der Newtonschritt nach (8.45) liefert mit (8.46) nach elementaren Rechnungen:

$$\begin{aligned} y_1 &= \frac{1}{2}\cdot\left(\widetilde{y}_0 + \frac{a}{\widetilde{y}_0}\right) = \frac{1}{2}\cdot\left\{\sqrt{a}\cdot(1+\varepsilon_0) + \frac{a}{\sqrt{a}\cdot(1+\varepsilon_0)}\right\} \in \mathbb{IR} \\ &= \sqrt{a}\cdot\left\{1 + \frac{\varepsilon_0^2}{2\cdot(1+\varepsilon_0)}\right\} = \sqrt{a}\cdot(1+\varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2\cdot(1+\varepsilon_0)} \end{aligned} \tag{8.47}$$

Eine Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers erhält man dann durch Intervallauswertung des obigen Quotienten für ε_{app}, wobei gilt: $|\varepsilon_0| \le \varepsilon(0)$. Ist dann $\widetilde{y}_1$ die auf der Maschine berechnete Näherung für y_1, so erhält man für den

[11] Wertet man in (8.45) den ersten Term aus, so erhält man einen relativen Fehler $4.44\ldots\cdot 10^{-16}$, der sich bei geschickter Auswertung des rechten Terms $g(\widetilde{x})$ noch halbieren lässt.

durch $\widetilde{y}_1 = \widetilde{g}(\widetilde{x}) = \sqrt{\widetilde{x}^2 - 1} \cdot (1 + \varepsilon_f)$ definierten relativen Gesamtfehler ε_f nach (7.1) von Seite 186 die Oberschranke:

$$\varepsilon(f) = \varepsilon(app) + \varepsilon(g) \cdot [1 + \varepsilon(app)], \quad |\varepsilon_f| \leq \varepsilon(f); \tag{8.48}$$

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

$$\sqrt{x^2-1} \approx g_4(x) := x - \frac{1}{2x}, \quad x \geq 44000 \tag{8.49}$$

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

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

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

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

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

$$\begin{aligned}
|\varepsilon_{app}| &= \left| \frac{\sqrt{x^2-1} - \left(x - \frac{1}{2x}\right)}{\sqrt{x^2-1}} \right| \\
&= \left| \frac{\left[\sqrt{x^2-1} - \left(x - \frac{1}{2x}\right)\right] \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]}{\sqrt{x^2-1} \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]} \right| \\
&= \frac{\frac{1}{4x^2}}{\sqrt{x^2-1} \cdot \left[\sqrt{x^2-1} + \left(x - \frac{1}{2x}\right)\right]} \\
|\varepsilon_{app}| &< \frac{1}{4x^2 \cdot (x^2-1)} =: \varepsilon(app) \qquad (8.50)
\end{aligned}$$

Der relative Gesamtfehler ε_{f_4} ist definiert durch

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

und für die Oberschranke $\varepsilon(f_4)$ erhält man nach (7.1) von Seite 186 die Beziehung:

$$\varepsilon(f_4) = \varepsilon(app) + \varepsilon(g_4) \cdot [1 + \varepsilon(app)]; \tag{8.51}$$

Die Berechnung der Fehlerschranke $\varepsilon(f_4)$ erfolgt mit dem Programm `sqrtx2m1_A4` auf Seite 308 .

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

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

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

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

$$\widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2 =: a, \quad \widetilde{x}, \delta \in S(2, 53), \quad a \in I\!R; \tag{8.52}$$

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

Führt man mit dem Startwert $\widetilde{y}_0$ nach (8.45) von Seite 295 einen Newtonschritt durch, so erhält man nach (8.47) die neue Näherung:

$$\begin{aligned} y_1 &= g_1(\delta) := \widetilde{y}_0 + \frac{1}{2} \cdot \frac{a - \widetilde{y}_0^2}{\widetilde{y}_0}, \quad a := \widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2, \\ &= \sqrt{a} \cdot (1 + \varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)}, \quad |\varepsilon_0| \leq \varepsilon(0), \end{aligned} \tag{8.53}$$

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

$$\widetilde{g}_1(\delta) := \widetilde{y}_0 \oplus 0.5 \odot [(2 \cdot \delta \oplus \delta \odot \delta) \ominus \widetilde{y}_0 \odot \widetilde{y}_0] \oslash \widetilde{y}_0 \in S(2, 53), \tag{8.54}$$

so erhält man für den durch

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

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

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

d.h. man erhält bei der Durchführung eines zusätzlichen Newton-Schritts bei der Auswertung von $g_1(\delta)$ nach (8.54) einen im Vergleich zu $\varepsilon(0) = 3.33\ldots \cdot 10^{-16}$ sehr viel größeren Auswertefehler. Der Grund hierfür ist jedoch offensichtlich:

- Wegen der kleinen Fehlerschranke $\varepsilon(0) = 3.33\ldots \cdot 10^{-16}$ ist der Startwert $\widetilde{y}_0$ schon so gut, dass das Newton-Verfahren quadratisch konvergiert.
- Daher muss der zweite Summand in (8.53) mit dem Faktor 1/2 als Korrekturglied im Vergleich zu $\widetilde{y}_0$ sehr klein ausfallen, so dass bei der Auswertung der Differenz $[(2 \cdot \delta \oplus \delta \odot \delta) - \widetilde{y}_0 \odot \widetilde{y}_0]$ nach (8.54) starke Auslöschung auftreten muss, die den großen relativen Fehler $\varepsilon(g_1)$ verursacht.

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

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

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

$$a - \widetilde{y}_0^2 = (2 \cdot \delta + \delta^2) - \widetilde{y}_0^2 = (2 \cdot \delta - s_1^2) + [\delta^2 - s_2 \cdot (\widetilde{y}_0 + s_1)] \tag{8.55}$$

und wegen $2 \cdot \delta \approx a \approx \widetilde{y}_0^2 \approx s_1^2$ folgt damit $a - \widetilde{y}_0^2 \approx (2 \cdot \delta - s_1^2)$, so dass diese Differenz rundungsfehlerfrei berechnet werden kann[12]:

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

In (8.55) haben wir damit den für die Fehlerabschätzung gewünschten Idealfall, dass zu einem rundungsfehlerfreien ersten Summanden (...) ein fehlerbehafteter aber betragsmäßig kleiner Summand [....] zu addieren ist, so dass jetzt ein nur kleiner Auswertefehler für den vollständigen Term

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

[12] Der erfahrene Numeriker wird dieses Ergebnis wegen $2 \cdot \delta \approx s_1^2$ zwar sofort akzeptieren, der eigentliche Beweis ergibt sich jedoch erst durch die gesicherte Fehlerabschätzung mit dem Programm `sqrtx2m1_A1.cpp` von Seite 300.

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

$$\begin{aligned} \widetilde{y}_1 &= \widetilde{g}_1(\delta) = g_1(\delta)\cdot(1+\varepsilon_{g_1}) = \sqrt{a}\cdot(1+\varepsilon_{app})\cdot(1+\varepsilon_{g_1}) \\ &= \sqrt{a}\cdot(1+\varepsilon_f), \quad |\varepsilon_{app}| \le \varepsilon(app), \quad |\varepsilon_{g_1}| \le \varepsilon(g_1), \end{aligned} \tag{8.56}$$

und für $|\varepsilon_f|$ berechnet sich dann eine Oberschranke $\varepsilon(f)$ nach (8.48) zu:

$$|\varepsilon_f| \le \varepsilon(f) := \varepsilon(app) + \varepsilon(g_1)\cdot[1+\varepsilon(app)]; \tag{8.57}$$

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

```
// Programm:  sqrtx2m1_A1.cpp;
// Berechnung der rel. Fehlerschranke eps(f) in A1:
// x = 1 + DEL;  --->  x^2-1 = 2*DEL + DEL^2;
// y0 = s1 + s2; y0 = sqrt(x^2-1) = sqrt(2*DEL + DEL^2) ist Startwert
// x - y0^2 = (2*DEL - s1^2) + [DEL^2 - s2*(y0 + s1)]
#include "abs_relh.hpp" // Wegen abs_mulh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen CuT26()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;
real T_x(const interval& Di, const real& delx)
{   // Di: Teilintervall von DEL
    interval Y0,B,C,D,E,S1,S2,G,H;
    real s1,s2,rel,app,delb,delc,dele,delh;
    int ex;

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

    s1 = Cut26(Inf(Y0));  s2 = Cut26(Sup(Y0));
```

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

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

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

    return rel;  // Schranke des relativen Gesamtfehlers.
}

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

Hinweise:

1. Die Funktion `Max_bnd_Xi(T_x,DEL,delx,diam,bnd)` unterteilt das übergebene Intervall `DEL` mittels `diam = 1e-6` in Teilintervalle `Di`, für die die Funktion `T_x(...)` die jeweilige relative Fehlerschranke bestimmt, deren Maximum als Gesamtfehlerschranke $\varepsilon(f) =$ `bnd` von `Max_bnd_Xi(...)` zurückgegeben wird. Weitere Informationen findet man auf Seite 75.

2. Für $\delta \in$ `Di` gilt für die entsprechenden Startwerte[13] $\widetilde{y}_0 \in$ `Y0`. Zerlegt man dann $\widetilde{y}_0 =$ `s1` $+$ `s2` mittels `s1 = Cut26(y0); s2 = y0 - s1;` so werden die positiven `s1`-Werte eingeschlossen durch `s1` $\in$ [`Cut26(Inf(Y0))`, `Cut26(Sup(Y0))`].

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

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_1 =$ [`succ(1.0)`$, 1 + 3 \cdot 2^{-12}]$ gilt für den in (8.56) definierten relativen Gesamtfehler ε_f nach (8.57) die Abschätzung:

$$\boxed{\left|\frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}}\right| \leq \varepsilon(f, A_1) = 2.221261 \cdot 10^{-16}} \tag{8.58}$$

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

In diesem Bereich liefert die Zerlegung $\widetilde{x}^2 - 1 = 2 \cdot \delta + \delta^2$ aus A_1 sehr viel größere Auswertefehler, da jetzt $\delta^2 \gg 2 \cdot \delta$ gilt und $\delta \odot \delta$ nicht rundungsfehlerfrei berechnet werden kann. Besser ist jetzt die rundungsfehlerfreie Zerlegung von $\widetilde{x} = x_1 + x_2$ durch die Anweisungen[14]: `x1 = Cut26(x); x2 = x - x1;` und man erhält:

$$a := \widetilde{x}^2 - 1 = (x_1^2 - 1) + x_2 \cdot (x + x_1) \in I\!R, \tag{8.59}$$

wobei jetzt der erste Summand $(x_1^2 - 1)$ rundungsfehlerfrei berechnet wird. Wegen der Beziehung $(x_1^2 - 1) \gg x_2 \cdot (x + x_1)$ wird dann bei der Auswertung der rechten Seite von (8.59) der relative Fehler nur wenig größer sein als die relative Schranke der Grundoperationen $\varepsilon(h) = 2^{-52} = 2.220..\cdot 10^{-16}$ bei nur hochgenauer Arithmetik. Mit der Maschinenzahl

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

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

$$y_1 := \frac{1}{2} \cdot \left(\widetilde{y}_0 + \frac{\widetilde{x}^2 - 1}{\widetilde{y}_0}\right) \equiv \widetilde{y}_0 + \frac{1}{2} \cdot \frac{(\widetilde{x}^2 - 1) - \widetilde{y}_0^2}{\widetilde{y}_0} =: g_2(\widetilde{x}) \in I\!R \tag{8.60}$$

Wie im vorherigen Abschnitt müssen wir jetzt darauf achten, dass der obige Zähler $(\widetilde{x}^2 - 1) - \widetilde{y}_0^2$ ohne Auslöschung ausgewertet werden kann. Deshalb zerlegen wir den

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

[14] Es gilt jetzt: $\widetilde{x} =$ `x`, $x_1 =$ `x1` und $x_2 =$ `x2`.

Startwert $\widetilde{y}_0 = s_1 + s_2$ durch die beiden Anweisungen $s_1 = \texttt{Cut26}(\widetilde{y}_0);\ s_2 = \widetilde{y}_0 - s_1;$ rundungsfehlerfrei in die beiden Summanden s_1, s_2, so dass der obige Zähler jetzt wie folgt geschrieben werden kann:

$$(\widetilde{x}^2 - 1) - \widetilde{y}_0^2 = [(x_1^2 - 1) - s_1^2] + \{x_2 \cdot (\widetilde{x} + x_1) - s_2 \cdot (\widetilde{y}_0 + s_1)\}, \tag{8.61}$$

wobei der erste Summand $[\,...\,]$ rundungsfehlerfrei berechnet wird und der betragsmäßig kleinere zweite Summand $\{\,...\,\}$ mit einem hinreichend kleinen Fehler behaftet ist. Wertet man jetzt mit (8.61) die Funktion $g_2(\widetilde{x})$ aus, so erhält man für den in $\widetilde{y}_1 := \widetilde{g}_2(\widetilde{x}) = g_2(\widetilde{x}) \cdot (1 + \varepsilon_{g_2})$ definierten relativen Fehler ε_{g_2} wie im vorhergehenden Abschnitt eine hinreichend kleine Oberschranke $\varepsilon(g_2)$.

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

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

Nach (8.53) ist dann der relative Approximationsfehler ε_{app} der verbesserten Näherung y_1 definiert durch:

$$y_1 = g_2(\widetilde{x}) = \sqrt{a} \cdot (1 + \varepsilon_{app}), \quad \varepsilon_{app} = \frac{\varepsilon_0^2}{2 \cdot (1 + \varepsilon_0)} \tag{8.62}$$

und eine Oberschranke $\varepsilon(app) \ge \varepsilon_{app}$ erhält man wieder durch Intervallauswertung des Quotienten für ε_{app} in (8.62), wobei $\varepsilon_0 \in [-\varepsilon(0), +\varepsilon(0)]$ zu berücksichtigen ist.

Die relative Gesamtfehlerschranke ε_f ist jetzt wieder definiert durch:

$$\begin{aligned} \widetilde{y}_1 &= \widetilde{g}_2(\widetilde{x}) = g_2(\widetilde{x}) \cdot (1 + \varepsilon_{g_2}) = \sqrt{a} \cdot (1 + \varepsilon_{app}) \cdot (1 + \varepsilon_{g_2}) \\ &= \sqrt{a} \cdot (1 + \varepsilon_f), \quad |\varepsilon_{app}| \le \varepsilon(app), \quad |\varepsilon_{g_2}| \le \varepsilon(g_2), \end{aligned} \tag{8.63}$$

und für $|\varepsilon_f|$ berechnet sich dann wieder eine Oberschranke $\varepsilon(f)$ nach (8.48) zu:

$$|\varepsilon_f| \le \varepsilon(f) := \varepsilon(app) + \varepsilon(g_2) \cdot [1 + \varepsilon(app)]; \tag{8.64}$$

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

```
// Programm:  sqrtx2m1_A2.cpp
// Berechnung des relativen Gesamtfehlers
// y0 +0.5*(x-y0^2)/y0;    x = x1+x2;   y0 = s1+s2;
// x-y0^2 = [(x1^2-1)-s1^2] + [x2*(x+x1)-s2*(y0+s1)]
#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen Cut26()
#include <iostream>     // Wegen cout
```

```
using namespace cxsc;
using namespace std;

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

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

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

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

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

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

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

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

Ergebnis:

Für alle Maschinenzahlen $\widetilde{x} \in A_2 = [1 + 3 \cdot 2^{-12}, 1024]$ gilt für den in Gleichung (8.63) definierten relativen Gesamtfehler ε_f nach (8.64) die Abschätzung:

$$\left| \frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2 - 1}}{\sqrt{\widetilde{x}^2 - 1}} \right| \leq \varepsilon(f, A_2) = 2.221305 \cdot 10^{-16} \tag{8.65}$$

Fehlerabschätzung in $A_3 = [1024, 44000]$

In diesem Bereich A_3 benutzen wir fast den gleichen Algorithmus wie in A_2, lediglich der in (8.61) angegebene Term

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

wird jetzt wie folgt umgeschrieben:

$$(\widetilde{x}^2 - 1) - \widetilde{y}_0^2 = [(x_1^2 - s_1^2) - 1] + \{x_2 \cdot (\widetilde{x} + x_1) - s_2 \cdot (\widetilde{y}_0 + s_1)\}, \tag{8.66}$$

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

```
// Programm:  sqrtx2m1_A3.cpp
// Berechnung des relativen Gesamtfehlers
// y0 +0.5*(x-y0^2)/y0
// x = x1+x2;   y0 = s1+s2;
// x-y0^2 = [(x1^2-s1^2)-1] + [x2*(x+x1)-s2*(y0+s1)]
#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen Cut26()
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

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

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

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

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

```
    S2 = interval(0,r1);  // s1 in S1 und s2 in S2 enthalten!!

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

    // Berücksichtigung des Approximationsfehlers:
    r1 = addu(1.0,app);
    r2 = mulu(rel,r1);
    rel = addu(app,r2);

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

int main()
{
    interval X = interval(1024,44000);
    real bnd, diam = 1e-6, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Ergebnis:

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

$$\boxed{\left|\frac{\widetilde{y}_1 - \sqrt{\widetilde{x}^2-1}}{\sqrt{\widetilde{x}^2-1}}\right| \leq \varepsilon(f, A_3) = 2.220461 \cdot 10^{-16}} \tag{8.67}$$

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

In diesem Bereich benutzen wir die Approximation:

$$\sqrt{x^2-1} \approx g_4(x) := x - \frac{1}{2x}, \quad x \geq 44000; \tag{8.68}$$

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

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

Da die rechte Seite monoton fällt, gilt die Abschätzung[15]:

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

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

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

und nach (7.1) von Seite 186 gilt für den relativen Gesamtfehler ε_f wieder die Abschätzung:

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

Die Berechnung der Oberschranke $\varepsilon(f)$ erfolgt mit dem nachfolgenden Programm `sqrtx2m1_A4.cpp`:

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

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

```
#include "abs_relh.hpp" // Wegen abs_divh1()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
using namespace cxsc;
using namespace std;

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

    abs_divh1(interval(1),0.0,Xi,delx,A,app);  // A = 1/Xi
    abs_divh1(A,app,interval(2),0.0,B,delb);   // B = [1/Xi]/2
    rel_subh1(Xi,0.0,B,delb,A,rela);           // A = Xi - [1/Xi]/2
    // rela: relativer Auswertefehler von x - [1/x]/2
    // Berechnung des relativen Approximationsfehlers:
    B = Xi*Xi;
    A = 1 / (4*B*(B-1));
    app = Sup(A);  // app:  relativer Approximationsfehler
    // Berücksichtigung des Approximationsfehlers:
    rel  = addu(1.0,app);
    delb = mulu(rel,rela);
    rel  = addu(app,delb);
    return rel; // Rückgabe einer Schranke rel des rel. Gesamtfehlers.
}
int main()
{
    interval X = interval(44000,1e10);
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X,delx,diam,bnd);
    cout << RndUp << "Relative Fehlerschranke = " << bnd << endl;
}
```

Ergebnis:

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

$$\left| \frac{\widetilde{g}_4(\widetilde{x}) - \sqrt{\widetilde{x}^2-1}}{\sqrt{\widetilde{x}^2-1}} \right| \leq \varepsilon(f, A_4) = 2.221114 \cdot 10^{-16} \tag{8.69}$$

Zusammenfassung:

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

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

Bei der vorausgesetzten hochgenauen Arithmetik ist die relative Fehlerschranke der Grundoperationen gegeben durch $\varepsilon(h) := 2^{-52} = 2.2204\ldots \cdot 10^{-16}$. Da die obige Schranke $\varepsilon(f)$ nur minimal größer ist als $\varepsilon(h)$, habe wir für $f(x) = \sqrt{x^2-1}$ bezüglich der Fehlerschranke einen nahezu optimalen Algorithmus gewählt. Lässt man etwas größere Fehlerschranken zu, so kann die Laufzeit der jeweiligen Implementierung nur geringfügig verbessert werden. Die Funktion $f(x) = \sqrt{\widetilde{x}^2-1}$ wird daher in **C-XSC** durch den in diesem Abschnitt beschriebenen Algorithmus realisiert, vgl. dazu auch den Quelltext auf Seite 312.

8.3.2 Intervallargumente

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

$$W_{\texttt{x}} := \left\{ y \in I\!R \,\middle|\, y = \sqrt{x^2-1} \ \wedge\ x \in \texttt{x} \right\} \subset \texttt{W} \in IR \tag{8.70}$$

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

Zur Berechnung von `Inf(W)` bestimmt man zunächst `r1=Sqrtx2m1(Inf(z))` und muss dann diesen Maschinenwert noch mit `q_sqrtx2m1m` < 1 multiplizieren, um eine garantierte Unterschranke aller Funktionswerte $f(x)$, mit $x \in \texttt{z}$ zu erhalten. Mit Hilfe der obigen Fehlerschranke $\varepsilon(f)$ wird `q_sqrtx2m1m` vorher durch Aufruf der Funktion `eps2fractions(...)` aus dem Modul `bnd_util` berechnet. Weitere Einzelheite dazu findet man auf Seite 81.

Die Berechnung von `Sup(W)` erfolgt ganz entsprechend. Man berechnet zunächst wieder `r2=Sqrtx2m1(Sup(z))` und muss dann noch abschließend mit der Konstanten `q_sqrtx2m1p` > 1 multiplizieren, um eine garantierte Oberschranke aller Funktionswerte $f(x)$, mit $x \in \texttt{z}$ zu erhalten. Im Falle $\texttt{expo}(\texttt{Sup}(\texttt{z})) >= 26$ ist jedoch `Sup(z)` wegen $\sqrt{x^2-1} < |x|$ die optimale Oberschranke, vgl. dazu Seite 312,... .

8.3.3 Numerische Ergebnisse

Punktargumente:

1. $\widetilde{x} = 1$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{0.00000000000000000000E+000}$
2. $\widetilde{x} = \texttt{succ(1.0)}$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{2.10734242554470173340E-008}$
3. $\widetilde{x} = 1.0009765625$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{4.42049621006104509480E-002}$
4. $\widetilde{x} = 1.03125$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{2.51945554634329660360E-001}$
5. $\widetilde{x} = 2$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.73205080756887719318E+000}$
6. $\widetilde{x} = 520$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{5.19990384606494444460E+002}$
7. $\widetilde{x} = 1024$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.02399951171863358468E+003}$
8. $\widetilde{x} = 1025$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.02499512195005763720E+003}$
9. $\widetilde{x} = 44000$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{4.39999999886363657424E+004}$
10. $\widetilde{x} = \texttt{MaxReal}$ $\rightsquigarrow$ $\texttt{Sqrtx2m1}(\widetilde{x}) = \texttt{1.79769313486231570815E+308}$

Intervallargumente:

1. $\texttt{x} = [1, 1]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{0.000000000000000E+000},\texttt{0.000000000000000E+000}]$
2. $\texttt{x} = [\texttt{succ(1.0)},\texttt{succ(1.0)}]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{2.107342425544700E-008},\texttt{2.107342425544705E-008}]$
3. $\texttt{x} = [1, 2]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{0.000000000000000E+000},\texttt{1.732050807568880E+000}]$
4. $\texttt{x} = [2, 2]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.732050807568876E+000},\texttt{1.732050807568880E+000}]$
5. $\texttt{x} = [1025, 1025]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.024999512195005E+003},\texttt{1.024999512195007E+003}]$
6. $\texttt{x} = [32345678, 32345678]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{3.234567799999996E+007},\texttt{3.234567800000003E+007}]$
7. $\texttt{x} = [42345678, 42345678]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{4.234567799999995E+007},\texttt{4.234567800000000E+007}]$
8. $\texttt{x} = [\texttt{MaxReal},\texttt{MaxReal}]$
   $\rightsquigarrow$ $W_{\texttt{x}} \subset \texttt{W} \subseteq [\texttt{1.797693134862314E+308},\texttt{1.797693134862316E+308}]$

8.3.4 Quelltext für Punkt- und Intervallargumente

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

using namespace std;
using namespace cxsc;

real q_sqrtx2m1p(4503599627370501.0/4503599627370496.0);  // (1+e(f))
real q_sqrtx2m1m(9007199254740986.0/9007199254740992.0);  // (1-e(f))

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

```
        x2 = x1*x1;  ep = x2-1;
        res = sqrt(ep+ep2); // res ist Startwert für Newton-Verfahren
        s1 = Cut26(res);  s2 = res - s1;  // Startwert = s1 + s2;
        ep2 = ep2 - s2 * (res+s1); // ep2 = [x2*(x+x1)-s2*(y0+s1)]
        if (arg<c3) ep -= s1*s1;   // ep = (x1^2-1) - s1^2;
        else {
            x2 -= s1*s1;  // x2 = x1^2-s1^2
            ep = x2 - 1; }         // ep = (x1^2-s1^2) - 1;
        ep += ep2;        // ep = x - y0^2;
        ep /= res;
        times2pown(ep,-1);
        res = res + ep;  // 1. Newton-Schritt in hoher Genauigkeit
                         // beendet;  eps = 2.221305E-16
    } else { // arg = |x| >= 44000;
        res = -1/arg;
        times2pown(res,-1); // Multiplikation mit 0.5;
        res += arg;  // res = x - 1/(2*x);  eps = 2.221114E-16
    }
    return res;
} // Sqrtx2m1 (Punktargumente)

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

int main()
{
    interval x,y;

    while (1) {
        cout << "interval x = ?" << endl;
```

```
        cin >> x;
        y = Sqrtx2m1(x);
        cout << SetPrecision(20,20) << Scientific << "Sqrtx2m1(x) = "
             << y << endl << endl;
    };
}
```

Hinweis:

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

8.4 $\sqrt{1-x^2}$

Als weiteres Beispiel zur Fehlerabschätzung betrachten wir die reelle Funktion

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

Für alle Maschinenzahlen[16] $0 \le \widetilde{x} \le \texttt{pred}(1.0)$ ist eine garantierte Oberschranke $\varepsilon(f)$ des relativen Fehlers

$$\left| \frac{\sqrt{1-\widetilde{x}^2} - \widetilde{f}(\widetilde{x})}{\sqrt{1-\widetilde{x}^2}} \right| \le \varepsilon(f), \quad \widetilde{x} \in S(2,53), \quad \widetilde{x} \le \texttt{pred}(1.0); \tag{8.71}$$

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

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

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

8.4.1 Punktargumente

Für alle Punktargumente $\widetilde{x} \in [0, \texttt{pred}(1.0)]$ betrachten wir die vier Teilbereiche:

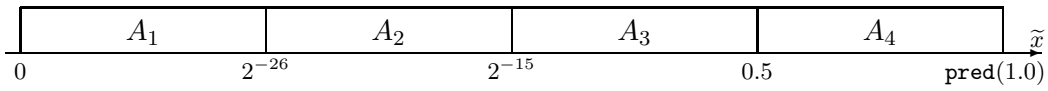


Abbildung 8.3: Teilintervalle A_i der Maschinenzahlen $\widetilde{x} \in S(2,53)$

In A_1 wird $f(x)$ approximiert durch $f(x) \approx 1$, in A_2 durch $f(x) \approx 1 - x^2/2$. In A_2, A_3 wird nach einer jeweils geeigneten Auswertung des Arguments $a = 1 - \widetilde{x}^2$ die Funktion $f(x)$ angenähert durch $\texttt{sqrt}(\widetilde{a}) \approx f(\widetilde{x})$, wobei $\texttt{sqrt}(\widetilde{a})$ die in **C-XSC** implementierte Wurzelfunktion ist, die hier mit dem fehlerbehafteten Maschinenargument $\widetilde{a} \approx 1 - \widetilde{x}^2$ aufgerufen wird. Wegen $f(-x) \equiv f(x)$ kann man sich stets auf $0 \le x \le \texttt{pred}(1.0) < 1$ beschränken.

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

Fehlerabschätzung in $A_1 = [0, 2^{-26})$

In diesem Teilbereich benutzen wir die laufzeitmäßig optimale Approximation

$$f(x) = \sqrt{1-x^2} \approx g_1(x) := 1$$

Der Betrag des relativen Approximationsfehlers $\varepsilon_{app} := (f(x) - g_1(x))/f(x)$ kann wie folgt abgeschätzt werden:

$$\begin{aligned} |\varepsilon_{app}| &= \left| \frac{\sqrt{1-x^2}-1}{\sqrt{1-x^2}} \right| = \left| \frac{\left[\sqrt{1-x^2}-1\right]\left[\sqrt{1-x^2}+1\right]}{\sqrt{1-x^2}\cdot\left[\sqrt{1-x^2}+1\right]} \right| \\ &= \frac{x^2}{(1-x^2)+\sqrt{1-x^2}} \leq \frac{x^2}{(1-x^2)+(1-x^2)} = \frac{x^2}{2\cdot(1-x^2)} =: \alpha_1(x) \end{aligned}$$

Da der letzte Term $\alpha_1(x)$ monoton in x wächst, erhält man in A_1 eine Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers, wenn man $\alpha_1(x)$ für $x = 2^{-26}$ intervallmäßig auswertet: $\alpha_1(2^{-26}) < 1.110224 \cdot 10^{-16} := \varepsilon(app)$.

$g_1(x) = 1$ lässt sich in $S(2,53)$ exakt darstellen, so dass der relative Auswertefehler verschwindet und der relative Gesamtfehler $\varepsilon(f, A_1)$ im Bereich A_1 nur durch den relativen Approximationsfehler definiert ist:

$$\left| \frac{\sqrt{1-\widetilde{x}^2}-1}{\sqrt{1-\widetilde{x}^2}} \right| \leq \varepsilon(f, A_1) = \varepsilon(app) = 1.110224 \cdot 10^{-16}, \quad \widetilde{x} \in A_1 \cap S(2,53)$$

Fehlerabschätzung in $A_2 = [2^{-26}, 2^{-15})$

In diesem Teilbereich benutzen wir die Approximation

$$f(x) = \sqrt{1-x^2} \approx g_2(x) := 1 - \frac{x^2}{2}$$

Der Betrag des relativen Approximationsfehlers $\varepsilon_{app} := (f(x) - g_2(x))/f(x)$ kann analog zum Bereich A_1 wie folgt abgeschätzt werden:

$$\begin{aligned} |\varepsilon_{app}| &= \left| \frac{\sqrt{1-x^2}-(1-\frac{x^2}{2})}{\sqrt{1-x^2}} \right| = \left| \frac{\left[\sqrt{1-x^2}-(1-\frac{x^2}{2})\right]\left[\sqrt{1-x^2}+(1-\frac{x^2}{2})\right]}{\sqrt{1-x^2}\cdot\left[\sqrt{1-x^2}+(1-\frac{x^2}{2})\right]} \right| \\ &= \frac{x^4/4}{(1-x^2)+\sqrt{1-x^2}\cdot(1-\frac{x^2}{2})} \leq \frac{x^4/4}{(1-x^2)+(1-x^2)} \\ &= \frac{x^4}{8\cdot(1-x^2)} =: \alpha_2(x) \end{aligned}$$

Da $\alpha_2(x)$ monoton wächst, erhält man für ein Teilintervall $\boldsymbol{x} = [\widetilde{x}_1, \widetilde{x}_2] \subset A_2$ eine Oberschranke $\varepsilon(app)$ des relativen Approximationsfehlers, wenn man die Funktion $\alpha_2(x)$ für $x = x_2$ intervallmäßig berechnet. Die Auswertung von $g_2(\widetilde{x})$ liefert das Maschinenergebnis $\widetilde{g}(\widetilde{x}) = 1 \ominus (\widetilde{x} \odot \widetilde{x})/2$, wobei für alle $\widetilde{x} \in \boldsymbol{x}$ die Division durch 2 rundungsfehlerfrei erfolgt. Eine Oberschranke $\varepsilon(g_2)$ des relativen Auswertefehlers $|(\widetilde{g}_2(\widetilde{x}) - g(\widetilde{x}))/g_2(\widetilde{x})| \leq \varepsilon(g_2)$ wird im Programm `sqrt1mx2_bnd.cpp` von Seite 320 in der Funktion `T_xA2(...)` durch `ra` $= \varepsilon(g_2)$ berechnet, und nach (7.1) von Seite 186 erhält man für den relativen Gesamtfehler die Abschätzung:

$$\left| \frac{\sqrt{1-\widetilde{x}^2} - g_2(\widetilde{x})}{\sqrt{1-\widetilde{x}^2}} \right| \leq \varepsilon(app) + \varepsilon(g_2) \cdot [1 + \varepsilon(app)] =: \varepsilon(f, \boldsymbol{x}) \quad \forall \widetilde{x} \in \boldsymbol{x}$$

Die Funktion `T_xA2` gibt $\varepsilon(f,\boldsymbol{x})$ mit der Variablen `rel` an die aufrufende Funktion `Max_bnd_Xi(T_xA2,A2,delx,diam,bnd2)` zurück und diese berechnet für alle Teilintervalle $\boldsymbol{x}$ mit `bnd2` das Maximum der einzelnen Schranken $\varepsilon(f,\boldsymbol{x})$, wobei der relative Durchmesser aller Teilintervalle $\boldsymbol{x}$ durch den Parameter `diam` gesteuert wird. Bei zu großen `diam`-Werten erhält man für die Schranken $\varepsilon(f,\boldsymbol{x})$ deutliche Überschätzungen. Mit `diam` $= 10^{-6}$ liefert das Programm `sqrt1mx2_bnd.cpp` das Ergebnis:

$$\left| \frac{\sqrt{1-\widetilde{x}^2} - g_2(\widetilde{x})}{\sqrt{1-\widetilde{x}^2}} \right| \leq 2.221531 \cdot 10^{-16} =: \varepsilon(f, A_2) \quad \forall \widetilde{x} \in A_2$$

Fehlerabschätzung in $A_3 = [2^{-15}, 0.5)$

In diesem Teilbereich wird das Argument $a := 1 - \widetilde{x}^2$ auf der Maschine direkt ausgewertet: $\widetilde{a} := 1 \ominus \widetilde{x} \odot \widetilde{x}$. Der dabei auftretende absolute Auswertefehler $\delta_a := \widetilde{a} - a$ bleibt dabei hinreichend klein, weil vom exakten Wert 1 ein nicht zu großer Wert $\widetilde{x} \odot \widetilde{x}$ subtrahiert wird, vgl. Sie dazu bitte die entsprechenden Anmerkungen auf Seite 295. Mit dem fehlerbehafteten Maschinenwert $\widetilde{a}$ wird dann die selbst fehlerbehaftete Wurzelfunktion `sqrt`$(\widetilde{a})$ aufgerufen. Eine Schranke $\varepsilon(f,\boldsymbol{x})$ des dabei auftretenden relativen Gesamtfehlers

$$\left| \frac{\texttt{sqrt}(\widetilde{a}) - \sqrt{1-\widetilde{x}^2}}{\sqrt{1-\widetilde{x}^2}} \right| \leq \varepsilon(f, \boldsymbol{x}) \quad \forall \widetilde{x} \in \boldsymbol{x} \subset A_3$$

kann dann für alle $\widetilde{x}$ aus einem Teilintervall $\boldsymbol{x} \subset A_3$ mit Hilfe der Funktion `T_xA3` aus dem gleichen **C-XSC** Programm `sqrt1mx2_bnd.cpp` auf Seite 320 berechnet werden. Der Rückgabewert `rel` $= \varepsilon(f,\boldsymbol{x})$ wird dabei intern an die aufrufende Funktion `Max_bnd_Xi(T_xA3,A3,delx,diam,bnd3)` übergeben, die für alle Teilintervalle $\boldsymbol{x} \subset A_3$ das Maximum dieser Fehlerschranken bestimmt und an die Variable `bnd3`

übergibt. Man erhält für die Schranke $\texttt{bnd3} = \varepsilon(f, A_3)$ des relativen Gesamtfehlers das Ergebnis:

$$\left|\frac{\texttt{sqrt}(\widetilde{a}) - \sqrt{1-\widetilde{x}^2}}{\sqrt{1-\widetilde{x}^2}}\right| \leq 3.700745 \cdot 10^{-16} =: \varepsilon(f, A_3) \quad \forall \widetilde{x} \in A_3$$

Fehlerabschätzung in $A_4 = [0.5, \texttt{pred}(1.0)]$

Wenn man in diesem Teilbereich das Argument $a := 1 - \widetilde{x}^2$ wie in A_3 direkt auswertet, so erhält man für $\widetilde{x} \to 1$ starke Auslöschung und damit einen zu großen Auswertefehler, der dann beim Aufruf der Wurzelfunktion in der Nähe des Ursprungs einen ebenfalls viel zu großen relativen Gesamtfehler produziert[17]. Der Fehler beim Auswerten von $a := 1 - \widetilde{x}^2$ kann durch den folgenden Algorithmus deutlich reduziert
werden:

Man berechnet zunächst die positive Differenz $d = 1 - \widetilde{x}$, die in diesem Bereich auf der Maschine exakt ausgewertet wird. Für das Argument a ergibt sich daraus:

$$a = 1 - \widetilde{x}^2 = 2 \cdot d - d^2, \qquad d = 1 - \widetilde{x} = 1 \ominus \widetilde{x} \quad \forall \widetilde{x} \in A_4$$

Die Auswertung auf der Maschine erfolgt durch: $\widetilde{a} := 2d \ominus d \odot d$, wobei die Multiplikation mit 2 rundungsfehlerfrei erfolgt. Damit haben wir wieder die für die Fehlerabschätzung günstige Situation, dass von einem exakt darstellbaren ersten Summanden $2d$ ein zwar fehlerbehafteter aber betragsmäßig kleiner zweiter Summand $d \odot d$ zu subtrahieren ist. Mit dem Maschinenwert $\widetilde{a}$ wird dann wieder die selbst fehlerbehaftete Wurzelfunktion $\texttt{sqrt}(\widetilde{a})$ aufgerufen, und die Fehlerabschätzung erfolgt mit dem Funktionsaufruf `Max_bnd_Xi(T_xA4,D,delx,diam,bnd4)` ganz analog zum Teilbereich A_3. Man erhält dann für die garantierte Schranke $\texttt{bnd4} = \varepsilon(f, A_4)$ des relativen Gesamtfehlers das Ergebnis:

$$\left|\frac{\texttt{sqrt}(\widetilde{a}) - \sqrt{1-\widetilde{x}^2}}{\sqrt{1-\widetilde{x}^2}}\right| \leq 3.700747 \cdot 10^{-16} =: \varepsilon(f, A_4) \quad \forall \widetilde{x} \in A_4$$

Im ganzen Bereich $0 \leq \widetilde{x} \leq 1$ ist dann die Schranke $\varepsilon(f)$ des relativen Gesamtfehlers das Maximum der vier Fehlerschranken $\varepsilon(f, A_i)$:

$$\boxed{\left|\frac{\texttt{sqrt}(\widetilde{a}) - \sqrt{1-\widetilde{x}^2}}{\sqrt{1-\widetilde{x}^2}}\right| \leq 3.700747 \cdot 10^{-16} =: \varepsilon(f) \quad \forall \widetilde{x} \in [-1, 1]}$$

[17] Beachten Sie, dass die Wurzelfunktion im Ursprung eine senkrechte Tangente besitzt, wodurch auch kleine Argumentfehler deutlich verstärkt werden.

Mit dem folgenden Programm `sqrt1mx2_bnd` können die einzelnen Fehlerschranken $\varepsilon(f, A_i)$ und $\varepsilon(f)$ automatisch berechnet werden:

```
// Programm:  sqrt1mx2_bnd.cpp;
// Berechnung der rel. Fehlerschranke eps(f) in
// 0 <= x <= pred(1.0) fuer sqrt( 1-x^2 );

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

using namespace cxsc;
using namespace std;

real T_xA2(const interval& x, const real& delx)
{   // x: Teilintervall von A2
    interval X2,R,z,zz;
    real delx2,ra,rel;

    abs_mulh1(x,delx,x,delx,X2,delx2);          // X2 = x*x
    abs_divh1(X2,delx2,interval(2),0.0,R,rel);  // R  = (x*x)/2
    rel_subh1(interval(1.0),0,R,rel,z,ra);      // z  = 1-(x*x)/2
    // ra: relativer Auswertefehler;
    z = Sup(X2);  zz = z*z;
    z = zz / (8*(1-z));  // Sup(z): relativer Approximationsfehler
    // Berechnung einer Schranke des relativen Gesamtfehlers:
    z = Sup(z);   zz = ra;
    z = z + zz*(1+z);
    rel = Sup(z);
    return rel;  // Schranke des relativen Gesamtfehlers.
}

real T_xA3(const interval& x, const real& delx)
{   // x: Teilintervall von A3
    interval X2,Az,R;
    real delx2,dela,rel;

    abs_mulh1(x,delx,x,delx,X2,delx2);  // X2 = x*x
    abs_subh1(interval(1.0),0,X2,delx2,Az,dela); // Az = 1-x^2
    rel_sqrt_abs(Az,dela,R,rel);

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

```
real T_xA4(const interval& Di, const real& delx)
{   // Di: Teilintervall von D
    interval G,H,C;
    real delh,delc,bnd;
    G=Di;
    times2pown(G,1);  // G = 2*Di
    abs_mulh1(Di,delx,Di,delx,H,delh);  // H = Di*Di
    abs_subh1(G,0,H,delh,C,delc);  // C = 2*Di - Di*Di
                  // delc: abs. Fehlerschranke bez. 2*Di - Di*Di
    rel_sqrt_abs(C,delc,G,bnd);    // G = sqrt(2*Di - Di*Di)
    return bnd;
}

int main()
{
    interval A1 = interval(0,comp(0.5,-25)),
             A2 = interval(comp(0.5,-25),comp(0.5,-14)),
             A3 = interval(0.1,0.5),
             A4 = interval(0.5,pred(1.0)),
             D  = 1 - A4,z,zz;
    real bnd1,bnd2,bnd3,bnd4,bnd=0,
         diam = 1e-6, delx=0;
    z = A1*A1;  z = Sup(z);
    z = z / (2*(1-z));
    bnd1 = Sup(z);
    if (bnd1>bnd) bnd = bnd1;
    cout << RndUp << "Rel. Schranke eps(f,A1) fuer A1  = " << bnd1
         << endl;
    Max_bnd_Xi(T_xA2,A2,delx,diam,bnd2);
    cout << "Rel. Schranke eps(f,A2) fuer A2  = " << bnd2 << endl;
    if (bnd2>bnd) bnd = bnd2;
    Max_bnd_Xi(T_xA3,A3,delx,diam,bnd3);
    cout << "Rel. Schranke eps(f,A3) fuer A3  = " << bnd3 << endl;
    if (bnd3>bnd) bnd = bnd3;
    Max_bnd_Xi(T_xA4,D,delx,diam,bnd4);
    cout << "Rel. Schranke eps(f,A4) fuer A4  = " << bnd4 << endl;
    if (bnd4>bnd) bnd = bnd4;
    cout << "Rel. Gesamtfehlerschranke eps(f) = " << bnd << endl;
}
```

Das Programm `sqrt1mx2_bnd` liefert die folgende Bildschirmausgabe:

```
Rel. Schranke eps(f,A1) fuer A1  = 1.110224E-016
Rel. Schranke eps(f,A2) fuer A2  = 2.221531E-016
Rel. Schranke eps(f,A3) fuer A3  = 3.700745E-016
Rel. Schranke eps(f,A4) fuer A4  = 3.700747E-016
Rel. Gesamtfehlerschranke eps(f) = 3.700747E-016
```

Hinweise:

1. In $A_4 = [0.5, \texttt{pred}(1.0)]$ könnte man das Argument $a = 1 - x^2$ auch auswerten als $a = (1 - x) \cdot (1 + x)$, zumal der erste Faktor $(1 - x)$ rundungsfehlerfrei berechnet wird. Mit dieser Produktdarstellung erhält man für den absoluten Auswertefehler $a - \widetilde{a}$ die Oberschranke $3.330672 \cdot 10^{-16}$, die jedoch deutlich größer ausfällt als die Oberschranke $2.220448 \cdot 10^{-16}$ bez. der gewählten Darstellung $a = 2d - d \cdot d$. Mit `sqrt1mx2_bnd.cpp` als Vorlage sollte der Leser die beiden Oberschranken mit einem eigenen Programm bestätigen.

2. In A_3, A_4 lässt sich die Fehlerschranke etwa auf den Wert $2.223... \cdot 10^{-16}$ noch weiter reduzieren, wenn man wie bei der Funktion $\sqrt{x^2 - 1}$ von Seite 294 mit Hilfe des schon berechneten Funktionswertes $\widetilde{y}_0$ zur Korrektur einen Newton-Schritt anschließt. Der verbesserte Werte $y_1 \in \mathbb{R}$ berechnet sich nach (8.45) durch

$$y_1 := \widetilde{y}_0 + \frac{1}{2} \cdot \frac{1 - \widetilde{x}^2 - \widetilde{y}_0^2}{\widetilde{y}_0}$$

   Eine wirkliche Verbesserung der Fehlerschranke erreicht man dabei aber nur, wenn der obige Zähler $z := 1 - \widetilde{x}^2 - \widetilde{y}_0^2$ hinreichend genau berechnet werden kann. Da $|z|$ bis zur Größenordnung 10^{-40} klein werden kann, ist eine Auswertung von z nur im *double*-Format sehr aufwendig, wobei $\widetilde{x}$ und $\widetilde{y}_0$ mit Hilfe der `Cut26`-Funktion mehrfach in Teilsummanden aufzuspalten sind und diese Teilsummanden in verschiedenen $\widetilde{x}$-Bereichen bei der notwendigen Simulation einer doppelten Genauigkeit in jeweils verschiedener Reihenfolge zu addieren sind. Numerische Untersuchungen haben gezeigt, dass der numerische Aufwand dabei fast genauso groß ist, wie bei der maximal genauen Auswertung von z mit Hilfe des Akkumulators. Da die Laufzeit durch den zusätzlichen Newtonschritt mindestens um den Faktor drei anwächst, wird bei der Implementierung der Funktion $\sqrt{1 - x^2}$ in **C-XSC** auf diese Korrektur verzichtet und die etwas größere relative Fehlerschranke $\varepsilon(f) := 3.700747 \cdot 10^{-16}$ akzeptiert. Sollte diese Genauigkeit in speziellen Fällen nicht ausreichen, so kann $\sqrt{1 - x^2}$ in einer Staggered Correction Arithmetik für die Datentypen `l_interval` oder `l_real` in **C-XSC** bis zu etwa 320 dezimalen Ziffern berechnet werden.

8.4.2 Intervallargumente

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

$$W_{\mathtt{x}} := \left\{ y \in I\!R \;\middle|\; y = \sqrt{1-x^2} \;\wedge\; x \in \mathtt{x} \right\} \subset \mathtt{W} \in IR \tag{8.72}$$

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

Zur Berechnung von `Inf(W)` bestimmt man zunächst `r1=sqrt1mx2(Sup(z))` und muss dann diesen Maschinenwert noch mit `q_sqrt1mx2m` < 1 multiplizieren, um eine garantierte Unterschranke aller Funktionswerte $f(x)$, mit $x \in \mathtt{z}$ zu erhalten. Mit Hilfe der Fehlerschranke $\varepsilon(f) = 3.700747 \cdot 10^{-16}$ wird `q_sqrt1mx2m` vorher durch Aufruf der Funktion `eps2fractions(...)` aus dem Modul `bnd_util` berechnet. Weitere Einzelheiten findet man auf Seite 81. Im Spezialfall $\mathtt{Sup}(\mathtt{z}) = 0$ wird $\mathtt{r1} = 1$ gesetzt.

Die Berechnung von `Sup(W)` erfolgt ganz entsprechend. Man bestimmt zunächst wieder `sqrt1mx2(Inf(z))` und muss dann noch mit `q_sqrt1mx2p` > 1 multiplizieren, um eine garantierte Oberschranke aller Funktionswerte $f(x)$, mit $x \in \mathtt{z}$ zu erhalten. Im Falle $\mathtt{Inf}(\mathtt{z}) < 4.81 \cdot 10^{-8}$ ist $\mathtt{r2} = 1$ wegen $\sqrt{1-x^2} \leq 1$ stets eine garantierte Oberschranke.

Bei Punktintervallen `z`, d.h. bei $\mathtt{Inf}(\mathtt{z}) = \mathtt{Sup}(\mathtt{z})$, wird aus Laufzeitgründen die Funktion `sqrt1mx2(...)` mit $\mathtt{r2} = \mathtt{sqrt1mx2}(\mathtt{Sup}(\mathtt{z}))$ nur einmal aufgerufen, vergleichen Sie dazu den Quelltext auf Seite 325.

8.4.3 Numerische Ergebnisse

Punktargumente:

1.	$\widetilde{x} = 1$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{0.00000000000000000000E+000}$
2.	$\widetilde{x} = 0$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{1.00000000000000000000E+000}$
3.	$\widetilde{x} = \mathtt{MinReal}$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{1.00000000000000000000E+000}$
4.	$\widetilde{x} = \mathtt{pred(1.0)}$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{1.49011611938476562500E-008}$
5.	$\widetilde{x} = 0.125$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{9.92156741649221518564E-001}$
6.	$\widetilde{x} = 0.25$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{9.68245836551854255347E-001}$
7.	$\widetilde{x} = 0.5$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{8.66025403784438596588E-001}$
8.	$\widetilde{x} = 0.75$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{6.61437827766147679043E-001}$
10.	$\widetilde{x} = 0.9375$	$\rightsquigarrow$	$\mathtt{sqrtx2m1}(\widetilde{x}) = \mathtt{3.47985272676876344899E-001}$

Intervallargumente:

1. $\mathtt{x} = [-1, -1]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,0.000000000000000E+000]
2. $\mathtt{x} = [0, 0]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.000000000000000E+000,1.000000000000000E+000]
3. $\mathtt{x} =$ [MinReal,MinReal]
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.999999999999992E-001,1.000000000000000E+000]
4. $\mathtt{x} =$ [pred(1.0),pred(1.0)]
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.490116119384764E-008,1.490116119384768E-008]
5. $\mathtt{x} = [0.125, 0.125]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.921567416492207E-001,9.921567416492227E-001]
6. $\mathtt{x} = [0.25, 0.25]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.682458365518534E-001,9.682458365518554E-001]
7. $\mathtt{x} = [0.5, 0.5]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [8.660254037844379E-001,8.660254037844396E-001]
8. $\mathtt{x} = [0.75, 0.75]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [6.614378277661471E-001,6.614378277661485E-001]
9. $\mathtt{x} = [0.9375, 0.9375]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.479852726768760E-001,3.479852726768768E-001]
10. $\mathtt{x} = [-1, +1]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,1.000000000000000E+000]
11. $\mathtt{x} = \langle 0.7001 \rangle$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [7.140448095182816E-001,7.140448095182831E-001]
12. $\mathtt{x} = \langle 0.1 \rangle$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.949874371066191E-001,9.949874371066211E-001]

Hinweise:

1. Nach (1.7) von Seite 8 gilt $\mathtt{pred}(1.0) = 1 - 2^{-53}$ und mit *Maple* erhält man: $\sqrt{1-(1-2^{-53})^2} = \sqrt{2^{-52} - 2^{-106}} = 1.49011611938476558364096 93... \cdot 10^{-8}$, d.h. mit 4. erhalten wir eine korrekte Einschließung des Funktionswertes.

2. In 12. bezeichnen wir mit $\boldsymbol{x} = \langle 0.1 \rangle$ dasjenige Intervall, das die im *double*-Format nicht darstellbare Zahl 0.1 optimal einschließt, d.h. es gilt $0.1 \in \boldsymbol{x}$ und $\mathtt{Inf}(\boldsymbol{x}) = \mathtt{pred}(\,\mathtt{Sup}(\boldsymbol{x})\,)$. Im Quelltext von Seite 325 erfolgt das Einlesen durch `cin >> x;` und die Tastatureingabe durch: $[0.1, 0.1]$ `<enter>`

8.4.4 Quelltext für Punkt- und Intervallargumente

```
// Programm sqrt1mx2.cpp zur Auswertung der Funktion
// sqrt(1-x^2) für Punktargumente x aus [-1,+1]
// und für Intervallargumente.

#include <rmath.hpp>  // Wegen sqrt()
#include <imath.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

real Sqrt1mx2(const real& x) throw(STD_FKT_OUT_OF_DEF)
// sqrt(1-x^2);  rel. Fehlerschranke:  eps = 3.700747E-16 = e(f)
{
    real t=x,res;
    int ex;
    if (sign(t)<0) t = -t; // Argument t >=0;
    if (t>1) cxscthrow(STD_FKT_OUT_OF_DEF
                                    ("real sqrt1mx2(const real&)"));
    // Fuer t gilt jetzt:  0 <= t <=1;
    ex = expo(t);
    if (ex<=-26) res = 1; // t < 2^(-26) --> res = 1
    else if (ex<=-15) {   // t < 2^(-15) --> res = 1-x^2/2
        res = t*t;
        times2pown(res,-1);
        res = 1 - res;
    } else {
        if (ex>=0)
        {  // ex>=0 --> t>=0.5;
            res = 1-t;   // res: delta = 1-t;
            t = res * res;
            times2pown(res,1); // res: 2*delta
            res = res - t;     // res: 1-x^2 = 2*delta - delta^2
        } else res = 1-t*t;    // res: Maschinenwert von 1-x^2
        res = sqrt(res);       // res: Nullte Naeherung
    }
    return res;
} // Sqrt1mx2
```

```
// Konstanten für die Intervallfunktion sqrt(1-x^2):
real q_sqrt1mx2p(4503599627370501.0/4503599627370496.0);  // (1+e(f))
real q_sqrt1mx2m(9007199254740985.0/9007199254740992.0);  // (1-e(f))

interval Sqrt1mx2(const interval& x)
// sqrt(1-x^2);  Blomquist, 13.04.04;
{
    interval z = abs(x);  // sqrt(1-t^2) monoton fallend in t aus z;
    real r1,r2,sz,iz;
    sz = Sup(z);  iz = Inf(z);
    // Berechnung der Unterschranke r1:
    r2 = sqrt1mx2(sz);
    r1 = (sz==0)? 1 : r2 * q_sqrt1mx2m;
    // Berechnung der Oberrschranke r2:
    if (iz<4.81e-8) r2 = 1; // r2=1 ist immer korrekte Oberschranke!
    else r2 = (sz==iz)? r2*q_sqrt1mx2p : sqrt1mx2(iz)*q_sqrt1mx2p;
    return interval(r1,r2);

} // Sqrtx2m1 (Intervallargumente)

int main()
{
    interval x,y;
    while (1) {
        cout << "interval x = ?" << endl;   cin >> x;
        y = Sqrt1mx2(x);
        cout << SetPrecision(15,15)
             << Scientific << "Sqrt1mx2(x) = " << y << endl << endl;
    };
}
```

Hinweise:

1. Zur obigen Intervalleingabe mit `cin >> x` beachten Sie bitte den Hinweis auf Seite 324.

2. Zur Vermeidung von Namenskonflikten beim Übersetzen des obigen **C-XSC** Programms `sqrt1mx2.cpp` wurden die Funktionen mit `Sqrt1mx2` bezeichnet. Die gleichen Funktionen sind im **C-XSC** System unter dem Namen `sqrt1mx2` deklariert, für Punktargumente in `rmath.hpp` und für Intervallargumente in `imath.hpp`.

8.5 e^{-x^2}

Als weiteres Beispiel zur Fehlerabschätzung betrachten wir die reelle Funktion

$$f : D_f = I\!R \to [0,1], \quad \text{mit} \quad f(x) = e^{-x^2}$$

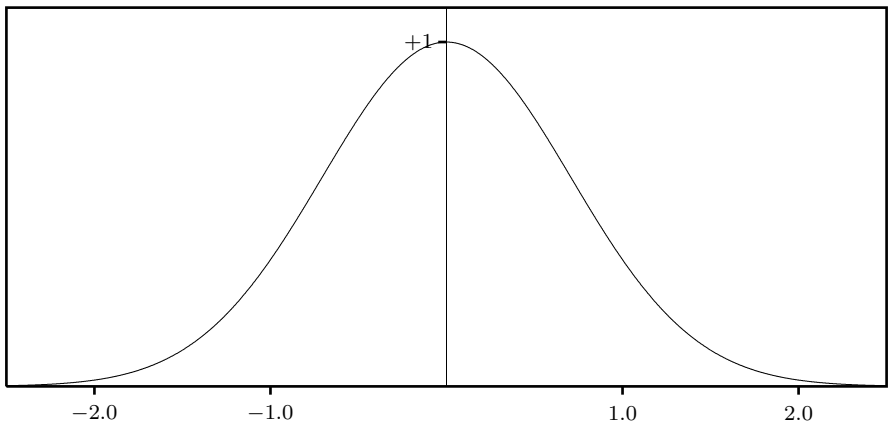


Abbildung 8.4: Die Funktion $f(x) = e^{-x^2}$

Für alle Maschinenzahlen $\widetilde{x}$, mit $0 \le |\widetilde{x}| \le x_0 \in S(2,53)$ ist eine garantierte und möglichst kleine Oberschranke $\varepsilon(f)$ des relativen Fehlers

$$\left| \frac{e^{-\widetilde{x}^2} - \widetilde{f}(\widetilde{x})}{e^{-\widetilde{x}^2}} \right| \le \varepsilon(f), \quad \widetilde{x} \in S(2,53), \quad 0 \le \widetilde{x} \le x_0; \tag{8.73}$$

zu berechnen, wobei $\widetilde{f}(\widetilde{x})$ die Maschinennäherung für den exakten Funktionswert $f(\widetilde{x}) = e^{-\widetilde{x}^2}$ ist. Auch jetzt wird wieder vorausgesetzt, dass bei der Berechnung von $\widetilde{f}(\widetilde{x})$ alle Grundoperationen in nur hochgenauer Arithmetik ausgeführt werden. Mit Hilfe von $\varepsilon(f)$ lässt sich dann zu einem vorgegebenen Maschinenintervall $X \subseteq D_f$ eine garantierte und nahezu optimale Einschließung $Y \in IR$ des Wertebereichs W_X berechnen:

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

$x_0 = 7491658466053896.0/281474976710656.0 \in S(2,53)$ ist die maximale Maschinenzahl, bis zu der mit Hilfe des Programms `expmx2_bnd.cpp` von Seite 332 die relative Fehlerschranke $\varepsilon(f)$ berechnet werden kann. Im Falle $\widetilde{x} > x_0 = 26.61571750\ldots$ werden bei der automatischen Fehlerabschätzung die Funktionswerte der intern benutzten `exp`-Funktion auf Null gesetzt, so dass ein relativer Fehler dann nicht mehr existiert.

Für alle $\widetilde{x}$, mit $|\widetilde{x}| > x_0$ gilt für die Maschinenwerte: $\widetilde{f}(\widetilde{x}) = 0$.

8.5.1 Punktargumente

Die Funktion $f(x) = e^{-x^2}$ wird auf der Maschine in vier Teilbereichen approximiert durch:

$$e^{-x^2} \approx \begin{cases} g_1(x) := 1 & \text{für } x < 2^{-26}, \quad \text{Bereich } A_1 \\ g_2(x) := 1 - x^2 + \frac{x^4}{2} - \frac{x^6}{6} & \text{für } x < 2^{-6}, \quad \text{Bereich } A_2 \\ g_3(x) := e^{-u} - v \cdot e^{-u}, x^2 = u + v & \text{für } x \le x_0, \quad \text{Bereich } A_3 \\ g_4(x) := 0 & \text{für } x > x_0, \quad \text{Bereich } A_4 \end{cases} \tag{8.74}$$

wobei man sich wegen $f(-x) \equiv f(x)$ auf $x \ge 0$ beschränken kann.

Fehlerabschätzung in $A_1 = [0, 2^{-26}]$

Da $g_1(x) \equiv 1$ exakt dargestellt wird, ist jetzt nur der relative Approximationsfehler abzuschätzen. Die Taylorreihe von $f(x)$ ist eine alternierende Leibnizreihe, so dass der absolute Approximationsfehler $|e^{-x^2} - 1|$ durch x^2 abgeschätzt werden kann. Für den relativen Approximationsfehler gilt dann

$$\left| \frac{e^{-x^2} - 1}{e^{-x^2}} \right| \le x^2 \cdot e^{x^2} =: h_1(x) \tag{8.75}$$

Da $h_1(x)$ monoton wächst, erhält man eine garantierte Oberschranke des relativen Approximationsfehlers, wenn man $h_1(x)$ für $x = 2^{-26}$ intervallmäßig auswertet und vom Ergebnisintervall das Supremum wählt. Im Programm `expmx2_bnd.cpp` von Seite 332 erfolgt dies in der Funktion `T_xA1` und man erhält:

$$\left| \frac{e^{-x^2} - 1}{e^{-x^2}} \right| \le x^2 \cdot e^{x^2} \le 2.220447 \cdot 10^{-16} = \varepsilon(f, A_1)$$

Fehlerabschätzung in $A_2 = [2^{-26}, 2^{-6}]$

Für $g_2(x)$ gilt nach dem Hornerschema:

$$\begin{aligned} g_2(x) &= 1 - x^2 + \frac{x^4}{2} - \frac{x^6}{6} \\ &= 1 - x^2 \cdot \left(1 - \frac{x^2}{2} \cdot \left(1 - \frac{x^2}{3}\right)\right) \end{aligned}$$

und die Auswertung auf der Maschine erfolgt mit $u := \widetilde{x} \odot \widetilde{x} \in S(2, 53)$ durch:

$$\widetilde{g}_2(\widetilde{x}) := 1 \ominus u \odot \left[1 - \frac{u}{2} \odot (1 \ominus u \oslash 3)\right]$$

wobei $u/2 = u \oslash 2$ exakt dargestellt wird. Im Programm `expmx2_bnd.cpp` wird der relative Auswertefehler für ein schmales Teilintervall `xi` in der Funktion `T_xA2(...)` mit der Variablen `rel_g2` berechnet.

Wie in (8.75) wird jetzt der relative Approximationsfehler abgeschätzt durch:

$$\left|\frac{e^{-x^2} - g_2(x)}{e^{-x^2}}\right| \le \frac{x^8}{24} \cdot e^{x^2} =: h_2(x) \tag{8.76}$$

Da $h_2(x)$ auch jetzt monoton wächst, erhält man im Teilintervall `xi` eine garantierte Oberschranke des relativen Approximationsfehlers, wenn man $h_2(x)$ für $x = \mathtt{Sup(xi)}$ intervallmäßig auswertet und vom Ergebnisintervall das Supremum bildet. Im Programm `expmx2_bnd.cpp` von Seite 332 wird dieser relative Approximationsfehler in der gleichen Funktion `T_xA2()` mit dem Intervall `z1` eingeschlossen. Bezeichnen wir hier für das Teilintervall `xi` den relativen Auswertefehler mit $\varepsilon(g_2)$ und den relativen Approximationsfehler mit $\varepsilon(app)$, so gilt nach (7.1) von Seite 186 für den relativen Gesamtfehler $\varepsilon(f, \mathtt{xi})$:

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_2(\widetilde{x})}{f(\widetilde{x})}\right| \le \varepsilon(f, \mathtt{xi}) = \varepsilon(app) + \varepsilon(g_2) \cdot [1 + \varepsilon(app)] \quad \forall\, \widetilde{x} \in \mathtt{xi} \subset A_2$$

In `T_xA2()` wird der obige Term für $\varepsilon(f, \mathtt{xi})$ intervallmäßig ausgewertet und durch `z1` eingeschlossen. Der relative Gesamtfehler wird dann für das Teilintervall `xi` als `Sup(z1)` zurückgegeben. Mit dem Funktionsaufruf

```
Max_bnd_Xi(T_xA2,A2,delx,diam,bnda)
```

erhält man dann für das Intervall $\mathtt{A2} = [2^{-26}, 2^{-6}]$ mit `bnda` das Maximum der Fehlerschranken $\varepsilon(f, \mathtt{xi})$ für alle Teilintervalle $\mathtt{xi} \subset \mathtt{A2} = A_2$. Der relative Durchmesser der `xi` wird durch `diam` gesteuert. Mit $\mathtt{diam} = 10^{-5}$ werden die Intervallüberschätzungen weitgehend vermieden, und man erhält i.a. eine ausreichende Genauigkeit. Da wir bei den Fehlerabschätzungen stets annehmen, dass die Funktionsargumente $\widetilde{x}$ rundungsfehlerfrei sind, gilt $\mathtt{delx} = 0$. Das Programm `expmx2_bnd.cpp` liefert mit dem obigen Funktionsaufruf für den Gesamtfehler im Intervall A_2 die Oberschranke

$$\left|\frac{f(\widetilde{x}) - \widetilde{g}_2(\widetilde{x})}{f(\widetilde{x})}\right| \le \mathtt{bnda} = \varepsilon(f, A_2) = 2.592509 \cdot 10^{-16} \quad \forall\, \widetilde{x} \in A_2$$

Beachten Sie bitte, dass $\varepsilon(f, A_2)$ sehr klein ausfällt, da der relative Approximationsfehler höchstens $h_2(2^{-6}) = 1.48\ldots \cdot 10^{-16}$ ist und weil bei der Berechnung von $\widetilde{g}_2(\widetilde{x})$ ein betragsmäßig nur kleiner fehlerbehafteter Wert vom exakten Wert 1.0 zu subtrahieren ist, wodurch nach Tabelle 3.1 auf Seite 59 vergleichsweise kleine Fehler erzeugt werden.

Fehlerabschätzung in $A_3 = [2^{-6}, x_0]$

Zur Fehlerabschätzung betrachten wir auch jetzt zunächst ein schmales Teilintervall $\texttt{xi} \subset A_3$. Für ein Maschinenargument $\widetilde{x} \in \texttt{xi}$ berechnet man zuerst mit Hilfe der Funktion `sqr2uv` die exakte Summe $\widetilde{x}^2 = u + v$ in doppelter Genauigkeit, wobei $u = \widetilde{x} \odot \widetilde{x} \approx \widetilde{x}^2$ das i.a. fehlerbehaftete Maschinenprodukt ist und $v \in S(2, 53)$ die betragsmäßig kleine Korrektur bedeutet. Mit $\texttt{xi2} := \texttt{xi} \odot \texttt{xi}$, $\texttt{Sxi2} := \texttt{Sup}(\texttt{xi} \odot \texttt{xi})$ und $\texttt{V} := \texttt{Sxi2} - \texttt{pred}(\texttt{Sxi2})$ gelten dann die Aussagen:

$$u \in \texttt{xi2}, \quad v \in [-\texttt{V}, +\texttt{V}] \quad \forall\, \widetilde{x} \in \texttt{xi}$$

$$e^{-\widetilde{x}^2} = e^{-u} \cdot e^{-v} = e^{-u} \cdot (1 - v + \delta_v) = e^{-u} - (v - \delta_v) \cdot e^{-u} \tag{8.77}$$

$f(\widetilde{x}) = e^{-\widetilde{x}^2}$ wird dann approximiert durch:

$$e^{-\widetilde{x}^2} \approx g_3(\widetilde{x}) := e^{-u} - v \cdot e^{-u} \equiv e^{-u} \cdot (1 - v) \tag{8.78}$$

und $g_3(\widetilde{x})$ wird auf der Maschine wie folgt ausgewertet:

$$\widetilde{g}_3(\widetilde{x}) := \texttt{exp}(-u) \ominus v \odot \texttt{exp}(-u) \tag{8.79}$$

Wählt man hier die in (8.78) rechts angegebene Produktdarstellung, so erhält man eine deutlich größere relative Fehlerschranke, da dann das Produkt $\texttt{exp}(-u) \odot (1 \ominus v)$ zweier fehlerbehafteter Faktoren zu berechnen ist. Im Gegensatz dazu wird in (8.79) wegen $|v| \ll u$ von einem fehlerbehafteten Wert ein ebenfalls fehlerbehafteter aber sehr viel kleinerer Maschinenwert $v \odot \texttt{exp}(-u)$ subtrahiert, womit ein viel kleinerer relativer Fehler verbunden ist. Dies gilt allerdings nur so lange, wie $v \odot \texttt{exp}(-u)$ im normalisierten Zahlenbereich liegt. Andernfalls wächst der absolute Fehler bei der Berechnung von $v \odot \texttt{exp}(-u)$ sprunghaft an, so dass dann die relative Fehlerschranke bei Auswertung von $\widetilde{g}(\widetilde{x})$ explosionsartig anwächst. Um dies zu vermeiden skaliert man $\texttt{exp}(-u)$ im Falle $v \neq 0$ rundungsfehlerfrei mit dem Faktor 2^{500}, berechnet also

$$\texttt{res} := 2^{500} \cdot \widetilde{g}(\widetilde{x}) = 2^{500} \cdot \texttt{exp}(-u) \ominus v \odot (2^{500} \cdot \texttt{exp}(-u))$$

womit sichergestellt ist, dass $v \odot (2^{500} \cdot \texttt{exp}(-u))$ nicht in den denormalisierten Bereich fallen kann, denn[18] für $\widetilde{x} >= 2^{-6}$ gilt $|v| \geq 2^{-12-104}$ und wegen $\widetilde{x} \leq x_0$ erhält man $\texttt{exp}(-u) > 2^{-1022}$, d.h. $|v| \odot (2^{500} \cdot \texttt{exp}(-u)) > 2^{-116+500-1022} = 2^{-638} > \texttt{MinReal}$.

Die Rückskalierung, d.h. die Multiplikation mit 2^{-500} erfolgt mit der Anweisung `times2pown(res,-500)` und liefert stets Funktionswerte $\texttt{res} > \texttt{MinReal} = 2^{-1022}$, so dass die Rückskalierung stets rundungsfehlerfrei erfolgt. Die Skalierung ist nicht

[18] $\widetilde{x} = 2^{ex}(1+2^{-52}) = \texttt{succ}(2^{ex}) \rightsquigarrow \widetilde{x}^2 = 2^{2 \cdot ex} \cdot (1+2^{-51}+2^{-104}) \rightsquigarrow u = 2^{2 \cdot ex}(1+2^{-51})$ und $v = 2^{2 \cdot ex - 104}$, damit gilt in A_3: $\widetilde{x} \geq 2^{ex} \implies v = 0$ oder $|v| \geq 2^{2 \cdot ex - 104} = 2^{-116} = 1.203\ldots \cdot 10^{-35}$;

nur im Funktionsalgorithmus sondern auch bei der automatischen Fehlerabschätzung anzuwenden, vgl. dazu das Programm `expmx2_bnd.cpp` auf Seite 332.

Nach (8.77) ist bei der Funktion $h_3(v) := v - \delta_v$ der absolute Approximationsfehler $\delta_v = e^{-v} - (1 - v)$ zu berücksichtigen. Wegen

$$\delta_v := \frac{v^2}{2} - \frac{v^3}{6} + \frac{v^4}{24} - \frac{v^5}{120} + - \ldots$$

gilt bei Anwendung der geometrischen Reihe die Abschätzung

$$|\delta_v| \le \frac{v^2}{2} \cdot \frac{1}{1 - |v|} \quad \forall\, |v| < 1$$

und für alle $\widetilde{x} \in \mathtt{xi}$ ist dann wegen $v \in [-\mathtt{V}, +\mathtt{V}]$

$$\Delta(\mathtt{V}) := \frac{\mathtt{V}^2}{2} \cdot \frac{1}{1 - \mathtt{V}}$$

eine Oberschranke des absoluten Approximationsfehlers, wobei $\Delta(\mathtt{V})$ wieder intervallmäßig auszuwerten ist und in der Funktion `T_xA3()` durch das Intervall `z` eingeschlossen wird. $\mathtt{r} = \mathtt{Sup(z)} \ge \Delta(\mathtt{V})$ ist dann die in der Funktion `T_xA3()` benutzte Oberschranke. Bei der Fehlerabschätzung benötigen wir jetzt noch eine garantierte Einschließung[19] der Funktionswerte $h_3(v) := v - \delta_v$. Wegen $v \in [-\mathtt{V}, +\mathtt{V}]$ und wegen $\delta_v \in [-\mathtt{r}, +\mathtt{r}]$ gilt damit: $h_3(v) \subset [-\mathtt{V}, +\mathtt{V}] + [-\mathtt{r}, +\mathtt{r}]$, wobei diese Intervallsumme in der Funktion `T_xA3()` durch das Maschinenintervall `vi` eingeschlossen wird.

Bei der Fehlerabschätzung benötigen wir zusätzlich noch für alle $\widetilde{x} \in \mathtt{xi}$ eine Einschließung aller Funktionswerte e^{-u}, und wegen $u \in \mathtt{xi2} := \mathtt{xi} \odot \mathtt{xi}$ ist diese Einschließung gegeben durch $e^{-u} \in \mathtt{exp}(-\mathtt{xi2})$, wobei dieses Maschinenintervall in der Funktion `T_xA3()` mit `exp_xi2` bezeichnet wird. Da wir e^{-u} auf der Maschine mit der fehlerbehafteten Funktion $\mathtt{exp}(-\mathtt{u}) \approx e^{-u}$ auswerten, brauchen wir zusätzlich auch noch eine Oberschranke des zugehörigen absoluten Approximationsfehlers:

$$\mathtt{exp}(-u) = e^{-u} \cdot (1 + \varepsilon_{exp}) = e^{-u} + \delta_u \quad \rightsquigarrow \quad \delta_u := \varepsilon_{exp} \cdot e^{-u}$$
$$|\delta_u| \le |\varepsilon_{exp}| \cdot \mathtt{Sup}(\mathtt{exp}(-\mathtt{xi} * \mathtt{xi})) \le 2.35796256 \cdot 10^{-16} \cdot \mathtt{Sup}(\mathtt{exp}(-\mathtt{xi} * \mathtt{xi}))$$

dabei wird die letzte Multiplikation mit Hilfe von `absF = mulu(c1,Sup(exp_xi2))` durchgeführt und das aufgerundete Ergebnis in `absF` gespeichert.

Mit dem Funktionsaufruf `Max_bnd_Xi(T_xA3,A3,delx,diam,bnda)` erhält man für den relativen Gesamtfehler im Intervall A_3 mit `bnda` die Oberschranke

$$\left| \frac{f(\widetilde{x}) - \widetilde{g}_3(\widetilde{x})}{f(\widetilde{x})} \right| \le \mathtt{bnda} = \varepsilon(f, A_3) = 4.618919 \cdot 10^{-16} \quad \forall\, \widetilde{x} \in A_3$$

[19] Beachten Sie bitte, dass der in diesem Buch benutzte Fehlerkalkül für jeden Operanden neben der absoluten oder relativen Fehlerschranke auch eine Einschließung der exakten Funktionswerte benötigt.

Zusammenfassung:

Im ganzen Bereich $0 \leq |\widetilde{x}| \leq x_0$ ist die Schranke $\varepsilon(f)$ des relativen Gesamtfehlers das Maximum der drei Fehlerschranken $\varepsilon(f, A_i)$:

$$\left|\frac{e^{-\widetilde{x}^2} - g_i(\widetilde{x})}{e^{-\widetilde{x}^2}}\right| \leq 4.618919 \cdot 10^{-16} =: \varepsilon(f) \quad \forall |\widetilde{x}| \leq x_0$$

Im Falle $|\widetilde{x}| > x_0 = 7491658466053896.0/281474976710656.0 \in S(2, 53)$ werden die Funktionswerte auf Null gesetzt; vgl. die Anmerkung auf Seite 341.

Mit dem folgenden Programm `expmx2_bnd` werden die einzelnen Fehlerschranken $\varepsilon(f, A_i)$ und $\varepsilon(f)$ automatisch berechnet:

```
// Programm:  expmx2_bnd.cpp;
// Berechnung der rel. Fehlerschranke eps(f) in
// 0 <= x <= x_0 = 26.61571750...  fuer e^(-x^2);
#include "abs_relh.hpp" // Wegen abs_mulh1(),...
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include "hilfe.hpp"    // Wegen sqr2uv(...)
#include <imath.hpp>    // Wegen exp(...)
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_xA1(const interval& A1)
{
    interval z;
    z = interval(Sup(A1));
    z = z*z;
    z = z*exp(z);
    return Sup(z);
}

real T_xA2(const interval& xi, const real& delx)
{   // e^(-x^2) = 1 - x^2*(1-x^2/2*[1-x^2/3])
    real delu,delz1,delz2,rel_g2,r;
    interval u,z1,z2;
    // Berechnung des Auswertefehlers von g2
    // nach dem Hornerschema:
```

```
    abs_mulh1(xi,0,xi,0,u,delu);       // u: Einschliessung von xi*xi
    abs_divh1(u,delu,interval(3),0,z1,delz1);          // z1: x^2/3;
    abs_1mx_delh(z1,delz1,z2,delz2);                  // z2: 1-x^2/3;
    abs_mulh1(u,delu,z2,delz2,z1,delz1);        // z1: x^2*(1-x^2/3);
    times2pown(z1,-1);  times2pown(delz1,-1); // z1: x^2/2*(1-x^2/3);
    abs_1mx_delh(z1,delz1,z2,delz2);      // z2: 1 - x^2/2*(1-x^2/3);
    abs_mulh1(u,delu,z2,delz2,z1,delz1);
                                      // z1: x^2*[1-x^2/2*(1-x^2/3)];
    rel_1mx_delh(z1,delz1,z2,rel_g2);
    // rel_g2: Rel. Auswertefehler von g_2(x)
    // Berechnung des Approximationsfehlers:
    z2 = interval(Sup(xi));  z2 = z2*z2;
    z2 = interval(Sup(z2));  // z2: x0^2
    z1 = z2*z2;  z1 = z1*z1; // z1: x0^8
    z1 = z1*exp(z2)/24;  // z1: Relativer Approximationsfehler;
    // Berechnung des relativen Gesamtfehlers:
    z2 = interval(rel_g2);
    z1 = z1 + z2*(1+z1);

    return Sup(z1);  // Rueckgabe des relativen Gesamtfehlers;
}

real T_xA3(const interval& xi, const real& delx)
{   // xi: Teilintervall von X; "e^(-u) + e^(-u)*(-v+delta)"
    // Mit Skalierung!
    const real c1 = 2.35796256e-16; // rel. Fehler von exp(...)
    interval xi2,exp_xi2,vi,z,S;
    real delh,dels,rel,absF,Sxi2,V,r;
    xi2 = xi*xi;  Sxi2 = Sup(xi2);
    exp_xi2 = exp(-xi2);          // e^(-u) in exp_xi2 enthalten
    times2pown(exp_xi2,1020);     // Skalierung mit 2^(1020)
    absF = mulu(c1,Sup(exp_xi2)); // absF: Abs. Fehler bez. e^(-u);
    V = subu(Sxi2,pred(Sxi2));
    vi = interval(-V,V);          // alle v in vi enthalten
    z = interval(V);
    z = (z*z/2)/(1-z);  r = Sup(z);  // r: abs. Approximationsfehler;
    vi = vi + interval(-r,r);        // vi: Einschliessung von:
                                     //     e^(-v)-1 = -v + delta
    abs_mulh1(exp_xi2,absF,vi,r,S,dels);    // S: e^(-u)*(-v+delta);
```

```
    rel_subh1(exp_xi2,absF,S,dels,z,rel);

    return rel;
}

int main()
{
    const real x0 = 7491658466053896.0 / 281474976710656.0;
    // x0 approx 26.61571750925125;    Die berechnete Fehlerschranke
    // e(f) = 4.618919E-16  gilt fuer alle Maschinenzahlen |x| <= x0

    interval
        A1 = interval(0,comp(0.5,-25)),
        A2 = interval(comp(0.5,-25),comp(0.5,-5)),
        A3 = interval(comp(0.5,-5),x0);
    real bnd,bnda, diam = 1e-5, delx=0;

    // Berechnung der relativen Fehlerschranken:
    bnd = T_xA1(A1);
    cout << "Bereich A1: Rel. Schranke eps(f,A1) = "
         << RndUp << bnd << endl;

    Max_bnd_Xi(T_xA2,A2,delx,diam,bnda);
    cout << "Bereich A2: Rel. Schranke eps(f,A2) = " << bnda << endl;
    if (bnda>bnd) bnd = bnda;

    Max_bnd_Xi(T_xA3,A3,delx,diam,bnda);
    cout << "Bereich A3: Rel. Schranke eps(f,A3) = " << bnda << endl;
    if (bnda>bnd) bnd = bnda;

    cout << "eps(f) = " << bnd << endl;
}
```

Das obige Programm liefert die folgende Bildschirmausgabe:

```
Bereich A1: Rel. Schranke eps(f,A1) = 2.220447E-016
Bereich A2: Rel. Schranke eps(f,A2) = 2.592509E-016
Bereich A3: Rel. Schranke eps(f,A3) = 4.618919E-016
eps(f) = 4.618919E-016
```

Hinweis zum Algorithmus:

Im Bereich A_3 könnte man das Newtonverfahren auf die Funktion $h(y) := \ln(y) + x^2$ anwenden und als Startwert für die Nullstellenbestimmung

$$h(y) = 0 \quad \Longleftrightarrow \quad y = e^{-x^2}$$

die Näherung $\widetilde{y}_0 := \texttt{exp}(-u) \in S(2,53)$ benutzen. Mit der bekannten Beziehung

$$y_1 := \widetilde{y}_0 - \frac{h(\widetilde{y}_0)}{h'(\widetilde{y}_0)}, \quad \widetilde{y}_0 \in S(2,53), \quad y_1 \in I\!R$$

erhält man die verbesserte Nullstellen-Näherung

$$\begin{aligned} y_1 &= \widetilde{y}_0 - \widetilde{y}_0 \cdot [\ln(\widetilde{y}_0) + x^2] = \widetilde{y}_0 - \widetilde{y}_0 \cdot [\ln(e^{-u} \cdot (1+\varepsilon_{exp})) + x^2] \\ &= \widetilde{y}_0 - \widetilde{y}_0 \cdot [-u + \ln(1+\varepsilon_{exp}) + (u+v)] \\ y_1 &= \widetilde{y}_0 - \widetilde{y}_0 \cdot [\ln(1+\varepsilon_{exp}) + v] \in I\!R \end{aligned} \tag{8.80}$$

Wie auf Seite 330 gilt: $v \in [-\texttt{V}, +\texttt{V}]$, und wegen[20] $\varepsilon_{exp} \in \texttt{z\_exp} := [-\varepsilon(exp), +\varepsilon(exp)]$ folgt: $[\ln(1+\varepsilon_{exp}) + v] \in [-\texttt{V}, +\texttt{V}] + \texttt{lnp1(z\_exp)}$. Da wir $[\ln(1+\varepsilon_{exp}) + v]$ auf der Maschine mit $v \in S(2,53)$ auswerten, ist damit eine garantierte Oberschranke des absoluten Approximationsfehlers gegeben durch $\texttt{S} := \texttt{Sup(abs(lnp1(z\_exp)))}$. Bitte beachten Sie, dass wir damit die rechte Seite von (8.80) wie folgt auswerten:

$$\widetilde{y}_1 := \widetilde{y}_0 \ominus v \odot \widetilde{y}_0, \quad \widetilde{y}_0 := \texttt{exp}(-u)$$

und damit den gleichen Algorithmus wie in (8.79) auf Seite 330 benutzen.

Da wir nach dem Newtonverfahren rechts in (8.80) die Maschinenzahl $\widetilde{y}_0$ als 0-te rundungsfehlerfreie Näherung betrachten, könnte man annehmen, dass die Fehlerabschätzung für die ganze Differenz eine Fehlerschranke liefert, die deutlich kleiner ist als der auf Seite 331 angegebene Wert $\varepsilon(f, A_3) = 4.618919 \cdot 10^{-16}$, da dort bei der Fehlerabschätzung $\widetilde{y}_0$ als fehlerbehaftet angenommen wurde. Erstaunlicherweise liefert jedoch eine Fehlerabschätzung zum Newtonverfahren mit Hilfe der Funktion

```
real T_xA3(const interval& xi, const real& delx)
{   // xi: Teilintervall von X; "e^(-u) + e^(-u)*(-v+delta)"
    // Mit Skalierung!
    const real c1 = 2.35796256e-16; // rel. Fehler von exp(...)
    interval xi2,exp_xi2,vi,T,z_exp,LNp1;
    real delh,dels,rel,absF,Sxi2,V,r,S;
```

[20] $\varepsilon(exp) = 2.35796256 \cdot 10^{-16}$ ist die relative Fehlerschranke der Exponentialfunktion.

```
    xi2 = xi*xi;  Sxi2 = Sup(xi2);
    exp_xi2 = exp(-xi2);          // e^(-u) in expmx2 enthalten
    times2pown(exp_xi2,500);  // Skalierung

    V = subu(Sxi2,pred(Sxi2));
    vi = interval(-V,V);          // alle v in vi enthalten
    z_exp = interval(-c1,c1);
    LNp1 = abs( lnp1(z_exp) );
    S = Sup(LNp1);  // S:  abs. Approximationsfehler
    vi = vi + LNp1; // vi: Einschließung von [ln(1+e_exp)+v]

    abs_mulh1(exp_xi2,0,vi,S,T,dels);  // T: e^(-u)*(v+ln(1+e_exp));
    rel_subh1(exp_xi2,0,T,dels,vi,rel);

    return rel;
}
```

genau die gleiche obige Fehlerschranke $\varepsilon(f, A_3) = 4.618919 \cdot 10^{-16}$. Für den gleichen Algorithmus nach (8.79) erhalten wir damit für zwei ganz verschiedene Ansätze zur Fehlerabschätzung genau die gleichen Fehlerschranken, was i.a. nicht zu erwarten ist. Bei der Fehlerabschätzung zum Newtonverfahren wurde der Approximationsfehler bez. $y_1 \approx e^{-x^2}$ noch nicht berücksichtig. Da jedoch der relative Fehler bez. $\widetilde{y}_0 \approx e^{-x^2}$ in der Größenordnung 10^{-15} liegt und das Newtonverfahren quadratisch konvergiert, wird der Approximationsfehler bez. $y_1 \approx e^{-x^2}$ die Größenordnung 10^{-30} besitzen und damit die Fehlerschranke $\varepsilon(f, A_3) = 4.618919 \cdot 10^{-16}$ nur minimal vergrößern!

8.5.2 Intervallargumente

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

$$W_{\mathtt{x}} := \left\{ y \in I\!R \,\middle|\, y = e^{-x^2} \ \wedge\ x \in \mathtt{x} \right\} \subset \mathtt{W} \in IR \tag{8.81}$$

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

Zur Berechnung einer Unterschranke $\mathtt{r1} \leq e^{-\mathtt{Sup(z)}^2}$ bestimmt man zunächst mit `Sz = Sup(z)` und `y = expmx2(Sz)` die Maschinennäherung $\mathtt{y} \approx e^{-\mathtt{Sup(z)}^2}$, und im Falle[21] `(Sz <= exp_x2_x0)` erhält man mit `r1 = y * q_exp_x2m;` die gesuchte

[21] $x0 = 7491658466053896.0/281474976710656.0 \in S(2, 53)$ wird im Quelltext von Seite 339 mit `epx_x2_x0` bezeichnet.

Unterschranke. Für $\texttt{Sup(z)} > x_0$ liefert `y = expmx2(Sz)` die Unterschranke $\texttt{r1} = 0$, und im Fall $\texttt{Sup(z)} = 0$ wird `r1` auf 1.0 gesetzt.

Die Berechnung einer garantierten Oberschranke $\texttt{r2} \geq e^{-\texttt{Inf(z)}^2}$ ergibt sich im Falle `(Iz <= exp_x2_x0)` ganz analog mit den Anweisungen `y = expmx2(Inf(z))` und `r2 = y * q_exp_x2p`. Sollte dabei $\texttt{r2} > 1$ auftreten, so wird $\texttt{r2} = 1.0$ gesetzt. Mit dem Algebra-System *Maple* findet man[22]

$$e^{-\texttt{succ}(x_0)^2} = 2.225073858507447856659\ldots \cdot 10^{-308} > \texttt{MinReal} = 2^{-1022}$$

Rundet man die obige Zeichenkette "`2.22507385850744785659E-308`" zur nächstgrößeren Rasterzahl, so erhält man die im Quelltext auf Seite 339 angegebene *real*-Konstante `exp_x2_UB`, und da e^{-x^2} für $x > 0$ monoton fällt, ist `exp_x2_UB` eine Oberschranke aller Funktionswerte e^{-x^2} für $x \geq \texttt{succ}(x_0)$.

Ist `z` ein Punktintervall, d.h. gilt $\texttt{Inf(z)} = \texttt{Sup(z)}$, so muss der Funktionswert $\texttt{expmx2(Inf(z))} = \texttt{expmx2(Sup(z))}$ natürlich nur einmal berechnet werden!

8.5.3 Numerische Ergebnisse

Punktargumente:

Die unten benutzte Symbolik $\widetilde{x} \doteq 0.1$ bedeutet, dass die Zeichenkette "0.1" mittels `cin >> RndNext >> x;` zur nächsten Rasterzahl $\widetilde{x} = \texttt{x}$ gerundet wird.

1.	$\widetilde{x} = 0$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{1.000000000000000E+000}$
2.	$\widetilde{x} = 1$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{3.678794411714423E-001}$
3.	$\widetilde{x} = 2^{-24}$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{9.999999999999964E-001}$
4.	$\widetilde{x} = 2^{-23}$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{9.999999999999858E-001}$
5.	$\widetilde{x} = 2^{-22}$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{9.999999999999432E-001}$
6.	$\widetilde{x} = x_0$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{2.225073858507869E-308}$
7.	$\widetilde{x} = \texttt{succ}(x_0)$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{0.000000000000000E+000}$
8.	$\widetilde{x} = 10^{+300}$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{0.000000000000000E+000}$
9.	$\widetilde{x} \doteq 0.1$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{9.900498337491681E-001}$
10.	$\widetilde{x} \doteq -10.16$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{1.478058178485540E-045}$
11.	$\widetilde{x} \doteq 4.123$	$\rightsquigarrow$	$\texttt{expmx2}(\widetilde{x}) = \texttt{4.143545175357494E-008}$

[22] dazu muss man in **C-XSC** zunächst $\texttt{succ}(x_0)$ mit `IEEE2frac_int(succ(x0),Z,N)` darstellen als $\texttt{succ}(x_0) = \texttt{Z/Z} \in S(2, 53)$, mit $Z = 7491658466053897.0$ und $N = 281474976710656.0$;

Intervallargumente:

1. $\mathtt{x} = [0, 0]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.000000000000000E+000,1.000000000000000E+000]
2. $\mathtt{x} = [1, 1]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.678794411714420E-001,3.678794411714429E-001]
3. $\mathtt{x} = [2^{-24}, 2^{-24}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.999999999999955E-001,9.999999999999978E-001]
4. $\mathtt{x} = [2^{-23}, 2^{-23}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.999999999999849E-001,9.999999999999872E-001]
5. $\mathtt{x} = [2^{-22}, 2^{-22}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.999999999999422E-001,9.999999999999445E-001]
6. $\mathtt{x} = [x_0, x_0]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [2.225073858507866E-308,2.225073858507872E-308]
7. $\mathtt{x} = [\mathtt{succ}(x_0), \mathtt{succ}(x_0)]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,2.225073858507448E-308]
8. $\mathtt{x} = \langle 10^{+300} \rangle$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,2.225073858507448E-308]
9. $\mathtt{x} = \langle 0.1 \rangle$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [9.900498337491672E-001,9.900498337491695E-001]
10. $\mathtt{x} = \langle -10.16 \rangle$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.478058178485539E-045,1.478058178485596E-045]
11. $\mathtt{x} = \langle 4.123 \rangle$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [4.143545175357490E-008,4.143545175357531E-008]
12. $\mathtt{x} = [0, 1]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.678794411714420E-001,1.000000000000000E+000]
13. $\mathtt{x} = [-10.16, 4.123]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.478058178485539E-045,1.000000000000000E+000]
14. $\mathtt{x} = [0.1, 4.123]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [4.143545175357490E-008,9.900498337491695E-001]
15. $\mathtt{x} = [-10^{300}, +10^{300}]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,1.000000000000000E+000]

8.5.4 Quelltext für Punkt- und Intervallargumente

```
// Programm expmx2.cpp zur Auswertung der Funktion
// e^(-x^2) fuer Punktargumente x aus [-MaxReal,+MaxReal]
// und fuer Intervallargumente.

#include <rmath.hpp>  // Wegen exp(...)
#include <imath.hpp>  // Wegen abs(x)
#include "hilfe.hpp"  // Wegen sqr2uv(...)
#include <iostream>   // Wegen cout ...

using namespace std;
using namespace cxsc;

// real-Konstante  exp_x2_x0  fuer die Funktion e^(-x^2);
// Diese Konstante wird auch fuer die Intervallfunktion benoetigt!
const real exp_x2_x0 = 7491658466053896.0 / 281474976710656.0;
// Fuer x > exp_x2_x0 werden die Funktionswerte auf Null gesetzt.
// Die relative Fehlerschranke e(f) := 4.618919E-16 gilt fuer
// alle |x| <= exp_x2_x0 = 26.61571750925....

real Expmx2(const real& x) throw()
// e^(-x^2);  rel. Fehlerschranke:  eps = 4.618919E-16 = e(f) gilt
// fuer alle |x| <= exp_x2_x0 = 26.61571750925....
// exp_x2_x0 = 7491658466053896.0 / 281474976710656.0;
// Fuer |x| > exp_x2_x0 --> expmx2(x) = 0;
// Ausfuehrlich getestet;  Blomquist, 05.07.04;
{
    real t=x,u,v,res=0;
    int ex;
    if (t<0) t = -t;  // t >= 0;
    ex = expo(t);
    if (ex<=-26) res = 1;  // t < 2^(-26)
    else if (ex<=-6)       // t < 2^(-6)
    {
        u = t*t;  v = u;
        times2pown(v,-1);  // v: 0.5*x^2
        res = 1-u*( 1-v*(1-u/3) );
    } else if (t<=exp_x2_x0) {
        sqr2uv(x,u,v);  // u:= x*x,v aus S(2,53); x^2 = u+v (exakt!)
        res = exp(-u);
```

```
        if (v!=0) {
            times2pown(res,500);  // Die Skalierung verhindert, dass
            v *= res; // v*exp(-u) in den denormal. Bereich faellt.
            res -= v;
            times2pown(res,-500); // Rueckskalierung
        }
    }
    return res;
} // expmx2

// Konstanten fuer die Intervallfunktion exp(-x^2):
real q_exp_x2p = 4503599627370502.0 / 4503599627370496.0; // (1+e(f))
real q_exp_x2m = 9007199254740984.0 / 9007199254740992.0; // (1-e(f))
// Oberschranke exp_x2_UB fuer |x| >exp_x2_x0:
real exp_x2_UB = 2.22507385850744856659E-308;

interval Expmx2(const interval& x)
// e^(-x^2);  Blomquist, 05.07.04;
{
    real y,r1,r2,Sz,Iz;
    interval z = abs(x);
    // Berechnung einer Unterschranke:
    Sz = Sup(z);  Iz = Inf(z);
    y = Expmx2(Sz);
    r1 = y;
    if (Sz<=exp_x2_x0) r1 = r1 * q_exp_x2m; // Abrunden
    if (Sz==0) r1 = y;
    // Berechnung einer Oberschranke:
    if (Iz>exp_x2_x0) r2 = exp_x2_UB;
    else r2 = (Sz==Iz)? y * q_exp_x2p : Expmx2(Iz) * q_exp_x2p;
    if (r2>1) r2 = 1.0;
    return interval(r1,r2);

} // Expmx2 (Intervallargumente)

int main() {
    real x,y;
    interval z,yz;
```

```
    while (1) {
        cout << "real x = ?" << endl;       cin >> RndNext >> x;
        y = Expmx2(x);
        cout << "Expmx2(x) = " << Scientific << SetPrecision(15,15)
             << y << endl;

        cout << "interval z = ?" << endl;   cin >> z;
        yz = Expmx2(z);
        cout << "Expmx2(z) = " << yz << endl << endl;
    };
}
```

Hinweise:

1. Mit den obigen Anweisungen

   ```
   cout << "real x = ?" << endl;  cin >> RndNext >> x;
   ```

   rundet man die Tastatureingabe zur nächsten *real*-Zahl und speichert diese in der Variablen `x`.

2. Zur obigen Intervalleingabe mit `cin >> z` beachten Sie bitte den Hinweis auf Seite 324.

3. Zur Vermeidung von Namenskonflikten beim Übersetzen des obigen **C-XSC** Programms `expmx2.cpp` wurden die Funktionen mit `Expmx2` bezeichnet. Die gleichen Funktionen sind in **C-XSC** unter dem Namen `expmx2` deklariert, für Punktargumente in `rmath.hpp` und für Intervallargumente in `imath.hpp`.

Abschließend noch eine Anmerkung zur relativen Fehlerschranke, die auf Seite 332 mit $\varepsilon(f) = 4.618919 \cdot 10^{-16}$ angegeben wurde. Man kann die Frage stellen, ob $\varepsilon(f)$ bei dem vorliegenden Algorithmus durch Intervallüberschätzungen oder durch zu grobe Werkzeuge bei der Berechnung der Fehlerschranke noch deutlich zu groß ausgefallen ist, oder ob $\varepsilon(f)$ optimal ist, d.h. ob die tatächlich auftretenden relativen Fehler nur wenig kleiner sind als $\varepsilon(f)$. Die Frage lässt sich nur beantworten, wenn man die tatsächlichen Fehler für möglichst viele Maschinen-Argumente $x \in [2^{-6}, x_0]$ berechnet, deren Maximum $\varepsilon(M, f)$ bestimmt und dann mit $\varepsilon(f)$ vergleicht. Dabei muss man $f(x) = e^{-x^2}$ zunächst in staggered Arithmetik in doppelter Präzision approximieren und mit dem Maschinenwert `expmx2(x)` vom Typ *real* vergleichen, wobei der Algorithmus von `expmx2` so umzuschreiben ist, dass die auftretenden Grundoperationen nur mit gerichteter Rundung erfolgen und dass die intern benutzten Maschinenwerte der Exponentialfunktion mit maximalem Fehler berechnet werden. Für das Intervall $[2^{-6}, 26.0]$ findet man $\varepsilon(M, f) = 4.567557 \cdot 10^{-16} < \varepsilon(f)$ und damit die Bestätigung, dass $\varepsilon(f)$ nahezu optimal ist.

8.6 arcosh$(1+x)$

Als weiteres Beispiel zur Fehlerabschätzung betrachten wir die reelle, monoton wachsende Funktion

$$f : D_f = \left\{ x \in I\!R \,\middle|\, x \geq 0 \right\} \to I\!R, \quad \text{mit} \quad f(x) = \operatorname{arcosh}(1+x)$$

Für alle Maschinenzahlen[23] $0 \leq \widetilde{x} \leq$ `MaxReal` ist eine garantierte Oberschranke $\varepsilon(f)$ des relativen Fehlers

$$\left| \frac{\operatorname{arcosh}(1+x) - \widetilde{f}(\widetilde{x})}{\operatorname{arcosh}(1+x)} \right| \leq \varepsilon(f), \quad \widetilde{x} \in S(2,53), \quad 0 \leq \widetilde{x} \leq \texttt{MaxReal}, \tag{8.82}$$

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

$$\left\{ y \in I\!R \,\middle|\, y = \operatorname{arcosh}(1+x),\ x \in X \right\} \subseteq Y$$

Beachten Sie bitte, dass wir mit Y eine Einschließung der Funktionswerte $y = f(x)$ auch für diejenigen Argumente $x \in X$ erhalten, die keine Maschinenargumente sind, obwohl sich die Fehlerschranke $\varepsilon(f)$ nur auf die Maschinenargumente $\widetilde{x} \in S(2,53)$ bezieht, da die Funktion f auf der Maschine nur für diese Argumente ausgewertet werden kann! $f(x) = \operatorname{arcosh}(1+x)$ wird z.B. bei der Implementierung der komplexwertigen $\arcsin(z)$-Funktion benötigt.

8.6.1 Algorithmen für arcosh$(1+x)$

In der Umgebung der Nullstelle $x = 0$ gilt die Darstellung:

$$\operatorname{arcosh}(1+x) = \sqrt{2x} \cdot Q(x) \tag{8.83}$$

$$= \sqrt{2x} \cdot \left(1 - \frac{1}{12}\, x + \frac{3}{160}\, x^2 \mp \ldots \right), \quad 0 \leq x < 2; \tag{8.84}$$

Für nicht zu große x gilt nach [1]

$$\operatorname{arcosh}(1+x) = \ln \left[1 + (x + \sqrt{2x + x^2}) \right], \tag{8.85}$$

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

und für $x \to \infty$ erhält man daraus:

$$\text{(8.86)} \qquad \operatorname{arcosh}(1+x) = \ln\left[1 + x \cdot (1 + \sqrt{1 + 2/x})\right], \quad x \to \infty$$

$$\text{(8.87)} \qquad \approx \ln(2x);$$

In (8.83) muss zunächst die Konvergenz der Reihe

$$Q(x) := \sum_{n=0}^{\infty} a_n \cdot x^n$$

näher untersucht werden. Differenziert man (8.83) nach x, so erhält man

$$\sqrt{1 + x/2} \cdot \left[2x \cdot Q'(x) + Q(x)\right] \equiv 1, \quad \text{und durch Einsetzen von } Q(x) \text{ folgt:}$$

$$\sqrt{1 + x/2} \cdot \sum_{n=0}^{\infty} (2n+1) \cdot a_n \cdot x^n = 1 \quad \text{bzw.}$$

$$\sum_{n=0}^{\infty} (2n+1) \cdot a_n \cdot x^n = \frac{1}{\sqrt{1 + x/2}} = \sum_{n=0}^{\infty} \binom{-0.5}{n} \cdot 2^{-n} \cdot x^n$$

und Koeffizientenvergleich liefert:

$$a_n = \frac{1}{(2n+1) \cdot 2^n} \cdot \binom{-0.5}{n}, \quad n = 0, 1, 2, \ldots$$

Mit dem Startwert $a_0 = 1$ ergibt sich daraus nach wirklich einfachen Umrechnungen die Rekursionsformel:

$$\text{(8.88)} \qquad a_{n+1} = -a_n \cdot \frac{(2n+1)^2}{4 \cdot (2n+3)(n+1)}, \quad n = 0, 1, 2, \ldots$$

Aus (8.88) ergibt sich direkt

$$\left| \frac{a_{n+1} \cdot x^{n+1}}{a_n \cdot x^n} \right| = x \cdot \frac{(2n+1)^2}{4 \cdot (2n+3)(n+1)} = x \cdot \frac{(2 + \frac{1}{n})^2}{4 \cdot (2 + \frac{3}{n})(1 + \frac{1}{n})} \xrightarrow[n \to \infty]{} \frac{x}{2}$$

und mit $q := x/2$ liefert dann das Quotientenkriterium unter der Voraussetzung $q < 1$, d.h. $0 \le x < 2$, dass $Q(x)$ für $0 \le x < 2$ sogar absolut konvergiert.

Für zukünftige Fehlerabschätzungen soll jetzt noch gezeigt werden, dass $Q(x)$ eine alternierende Leibniz-Reihe ist. Das Alternieren der Reihenglieder ergibt sich dabei wegen $0 \le x < 2$ direkt aus (8.88). Bleibt also zu zeigen:

$$\left| \frac{a_{n+1} \cdot x^{n+1}}{a_n \cdot x^n} \right| < 1 \quad \text{für alle } n = 0, 1, 2, \ldots ;$$

Nach (8.88) gilt

$$\begin{aligned}\left|\frac{a_{n+1}\cdot x^{n+1}}{a_n\cdot x^n}\right| &= \left|\frac{a_{n+1}}{a_n}\right|\cdot x = \frac{(2n+1)^2\cdot x}{4\cdot(2n+3)(n+1)} < \frac{(2n+1)^2\cdot x}{4\cdot(2n+1)(n+1)}\\ &= \frac{(2n+1)\cdot x}{4\cdot(n+1)} < \frac{(2n+2)\cdot x}{4\cdot(n+1)} = \frac{x}{2}\end{aligned}$$

Für die alternierende Leibniz-Reihe muss gelten $x/2 < 1$, und diese Bedingung ist im Konvergenzbereich von $Q(x)$ genau erfüllt.

In der nachfolgenden Tabelle sind die ersten zehn rationalen Koeffizienten a_n zusammengestellt.

Rationale $a_n = z/n$	$z \in S(2,53)$	$n \in S(2,53)$
a_0	$+1.0$	1.0
a_1	-1.0	12.0
a_2	$+3.0$	160.0
a_3	-5.0	896.0
a_4	$+35.0$	18432.0
a_5	-63.0	90112.0
a_6	$+231.0$	851968.0
a_7	-143.0	1310720.0
a_8	$+6435.0$	142606336.0
a_9	-12155.0	637534208.0

Tabelle 8.5: Ganzzahlige Werte $z, n \in S(2,53)$, mit $a_n = z/n$;

Nur der Wert $a_0 = 1.0$ kann in $S(2,53)$ exakt dargestellt werden.

8.6.2 Abschätzung der Approximationsfehler

Für $x \to 0$ benutzen wir nach (8.83) die Approximation

$$\operatorname{arcosh}(1+x) \approx \sqrt{2x}\cdot Q_N(x), \quad Q_N(x) := \sum_{n=0}^{N} a_n\cdot x^n;$$

Der Betrag des relativen Approximationsfehlers ε_{app} ist dann definiert durch:

$$|\varepsilon_{app}| := \left|\frac{\operatorname{arcosh}(1+x)-\sqrt{2x}\cdot Q_N(x)}{\operatorname{arcosh}(1+x)}\right| = \frac{\left|\sum_{n=N+1}^{\infty} a_n\cdot x^n\right|}{\left|\sum_{n=0}^{\infty} a_n\cdot x^n\right|} = \frac{A}{B}$$

Da $Q(x)$ im Konvergenzbereich $0 \le x < 2$ eine alternierende Leibnizreihe ist, gelten die folgenden Abschätzungen: $A \le |a_{N+1}|\cdot x^{N+1}$, $B \ge 1 - x/12$ $\rightsquigarrow$

$$|\varepsilon_{app}| \le \frac{|a_{N+1}|\cdot x^{N+1}}{1-x/12} = \varepsilon(app,N,x), \quad N = 0,1,2,\ldots \tag{8.89}$$

Für $x \to +\infty$ benutzen wir nach (8.87) die Approximation

$$\ln\left[1+x\cdot(1+\sqrt{1+2/x})\right] \approx \ln(2x), \quad x \to +\infty$$

Der relative Approximationsfehler ist dann definiert durch

$$\begin{aligned}\varepsilon_{app}(x) &:= \frac{\ln\left[1+x\cdot(1+\sqrt{1+2/x})\right]-\ln(2x)}{\operatorname{arcosh}(1+x)} = \frac{\ln\dfrac{1+x\cdot(1+\sqrt{1+2/x})}{2x}}{\operatorname{arcosh}(1+x)}\\ &= \frac{\ln(1+\alpha)}{\operatorname{arcosh}(1+x)}, \quad \text{mit}\end{aligned}$$

$$\begin{aligned}\alpha &:= \frac{1+x\cdot(1+\sqrt{1+2/x})-2x}{2x} = \frac{1-x+x\sqrt{1+2/x}}{2x}\\ &= \frac{1}{2x}+\frac{-1+\sqrt{1+2/x}}{2} = \frac{1}{2x}+\frac{1}{2x}-\frac{1}{16}\left(\frac{2}{x}\right)^2 \pm \ldots < \frac{1}{x}\end{aligned}$$

Dabei folgt die letzte Abschätzung aus der Tatsache, dass die Funktion $\sqrt{1+2/x}-1$ für $x \to +\infty$ selbst wieder eine alternierende Leibnizreihe ist. Da die ln-Funktion monoton wächst, erhalten wir

$$\begin{aligned}\varepsilon_{app}(x) &< \frac{\ln(1+\frac{1}{x})}{\operatorname{arcosh}(1+x)} < \frac{\ln(1+\frac{1}{x})}{\operatorname{arcosh}(x)} = \frac{\frac{1}{x}-\frac{1}{2}\cdot\left(\frac{1}{x}\right)^2 \pm \ldots}{\operatorname{arcosh}(x)}\\ &< \frac{1}{x\cdot\operatorname{arcosh}(x)} =: \varepsilon(app,x)\end{aligned} \tag{8.90}$$

Auch hier benutzen wir bei der letzten Abschätzung die Tatsache, dass die Reihe für $\ln(1+1/x)$ bez. $x \to +\infty$ eine alternierende Leibnizreihe ist.

8.6.3 Punktargumente

Für alle Punktargumente $\widetilde{x} \in [0, \texttt{MaxReal}]$ betrachten wir die vier Teilbereiche:

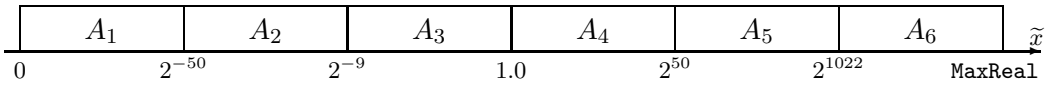


Abbildung 8.5: Teilintervalle A_i der Maschinenzahlen $\widetilde{x} \in S(2, 53)$

In A_1 wird $f(x)$ approximiert durch $f(x) \approx \sqrt{2x}$, in A_2 durch $f(x) \approx \sqrt{2x} \cdot Q_4(x)$. In A_3 gilt: $\operatorname{arcosh}(1+x) = \ln[1 + (x + \sqrt{2x + x^2})]$. In A_4 wählen wir $\operatorname{arcosh}(1+x) = \ln[1 + x \cdot (1 + \sqrt{1 + 2/x})]$. In A_5 wird die Approximation $\operatorname{arcosh}(1 + x) \approx \ln(2x)$ benutzt und in A_6 die Approximation $\operatorname{arcosh}(1 + x) \approx \ln(2) + \ln(x)$, um bei der Berechnung von $2 \cdot x$ einen Overflow zu vermeiden.

8.6.3.1 Fehlerabschätzung in A_1

In $A_1 = [0, 2^{-50})$ benutzen wir die Approximation

$$\operatorname{arcosh}(1 + x) \approx \sqrt{2x}\,, \quad 0 \le x < 2^{-50}$$

Nach (8.89) erhält man mit $N = 0$ die Oberschranke des in x monoton wachsenden relativen Approximationsfehlers

$$|\varepsilon_{app}| \le \varepsilon(app, 0, 2^{-50}) = \frac{2^{-50}}{12 - 2^{-50}} < \varepsilon(app, A_1) := 7.401487 \cdot 10^{-17}$$

Da das Argument $2x$ der Wurzelfunktion stets rundungsfehlerfrei berechnet wird, ist der relative Auswertefehler bei der Berechnung von $\sqrt{2x}$ gegeben durch den relativen Fehler der Wurzelfunktion $\varepsilon(sqrt) = 2.220447 \cdot 10^{-16}$, vgl. die Tabelle C.1 auf Seite 415. Der relative Gesamtfehler berechnet sich dann nach (7.1) von Seite 186

$$\varepsilon(f, A_1) := \varepsilon(app, A_1) + \varepsilon(sqrt) \cdot [1 + \varepsilon(app, A_1)] < \varepsilon(A_1) := 2.960596 \cdot 10^{-16};$$

Dabei entsteht $\varepsilon(A_1)$ durch Intervallauswertung des Ausdrucks für $\varepsilon(f, A_1)$.

8.6.3.2 Fehlerabschätzung in A_2

In $A_2 = [2^{-50}, 2^{-9})$ benutzen wir die Approximation

$$\operatorname{arcosh}(1 + x) \approx \sqrt{2x} \cdot Q_4(x), \quad 2^{-50} \le x < 2^{-9},$$

wobei das Polynom $Q_4(x)$ nach dem Hornerschema auszuwerten ist:

$$Q_4(x) := (((a_4 \cdot x + a_3) \cdot x + a_2) \cdot x + a_1) \cdot x + a_0,$$

dabei besteht zunächst das Problem, die in $S(2,53)$ nicht exakt darstellbaren Koeffizienten $a_1, \ldots, a_4$ unabhängig vom Rundungsmodus des Prozessors zur jeweils nächsten Rasterzahl zu runden, wobei Zähler und Nenner der Quotienten der a_j der Tabelle 8.5 auf Seite 344 entnommen werden können. Wir lösen das Problem durch Aufruf der Funktion

```
frac_tools(z,n,z,Next,del,eps);
```

aus dem Modul `bnd_util`, vgl. Seite 88. Hat man dann für $a_j = z/n$ mit `Next` vom Typ `real` die nächstgelegene Rasterzahl gefunden, so wird diese mit Hilfe der Funktion

```
void IEEE2frac_int(const real& Next, real& Z, real& N);
```

aus dem Modul `bnd_util` exakt darstellbar als Quotient `Next` = `Z`/`N`, wobei die Werte `Z,N` in $S(2,53)$ exakt gespeichert werden können. In nachfolgender Tabelle sind für die zur nächsten Rasterzahl gerundeten Koeffizienten $\dot{a}_j = Z/N$ die jeweiligen Werte $Z, N \in S(2,53)$ zusammengestellt:

$a_j = Z/N$	$Z \in S(2,53)$	$N \in S(2,53)$
a_0	$+1.0$	1.0
$\dot{a}_1$	-6004799503160661.0	72057594037927936.0
$\dot{a}_2$	$+5404319552844595.0$	288230376151711744.0
$\dot{a}_3$	-6433713753386423.0	1152921504606846976.0
$\dot{a}_4$	$+8756999275442631.0$	4611686018427387904.0

Tabelle 8.6: Ganzzahlige Werte $Z, N \in S(2,53)$, mit $\dot{a}_j = Z/N \neq a_j$;

Nur der Wert $a_0 = 1.0$ kann in $S(2,53)$ exakt dargestellt werden. Für die Fehlerabschätzung gilt folgendes

$$\begin{aligned}
\widetilde{\sqrt{2x}} \odot \widetilde{Q}_4(x) &= \sqrt{2x} \cdot (1+\varepsilon_{sqrt}) \odot Q_4(x) \cdot (1+\varepsilon_Q) \\
&= \sqrt{2x} \cdot Q_4(x) \cdot (1+\varepsilon_{sqrt}) \cdot (1+\varepsilon_Q) \cdot (1+\varepsilon_h) \\
&= \operatorname{arcosh}(1+x) \cdot (1+\varepsilon_{app}) \cdot (1+\varepsilon_{sqrt}) \cdot (1+\varepsilon_Q) \cdot (1+\varepsilon_h) \\
&= \operatorname{arcosh}(1+x) \cdot (1+\varepsilon_{A_2}),\ |\varepsilon_{sqrt}| \leq \varepsilon(sqrt), \quad |\varepsilon_h| \leq \varepsilon(h) = 2^{-52};
\end{aligned}$$

Wir benötigen jetzt also nur noch Oberschranken für $|\varepsilon_{app}|$ und $|\varepsilon_Q|$, um mit Hilfe der Funktion `eps_4()` aus Tabelle 5.1 von Seite 146 eine Oberschranke für $|\varepsilon_{A_2}|$ berechnen zu können. Wir beginnen mit der Abschätzung von $|\varepsilon_{app}|$ nach (8.89). Da $\varepsilon(app,4,x)$ monoton in x wächst, muss der Term

$$\varepsilon(app,4,2^{-9}) := \frac{|a_5| \cdot 2^{-45}}{1-2^{-9}/12} = \frac{63 \cdot 2^{-45}}{90112 \cdot (1-2^{-11}/3)} = \frac{63 \cdot 2^{-58}}{11 \cdot (1-2^{-11}/3)}$$

intervallmäßig ausgewertet werden. Man erhält das Ergebnis:

$$\varepsilon(app, 4, 2^{-9}) \leq \varepsilon(app) := 1.987371 \cdot 10^{-17}, \quad x \in A_2$$

Die Abschätzung des Auswertefehlers $|\varepsilon_Q|$ von $Q_4(x)$ erfolgt mit Hilfe des **C-XSC** Programms `Poly1_relh.cpp` von Seite 96, dabei werden die Rundungsfehler beim Runden der Polynomkoeffizienten a_j zur jeweils nächsten Rasterzahl automatisch berücksichtigt. Mit den Eingabewerten:

```
Polynomgrad N >= 1  ? 4   <enter>

u = x^k, 1<= k <=6  ? 1   <enter>

Zähler von a[1] = ? -1    <enter>
Nenner von a[1] = ? 12    <enter>

Zähler von a[2] = ? 3     <enter>
Nenner von a[2] = ? 160   <enter>

Zähler von a[3] = ? -5    <enter>
Nenner von a[3] = ? 896   <enter>

Zähler von a[4] = ? 35    <enter>
Nenner von a[4] = ? 18432 <enter>

Intervall der exakten Polynom-Argumente [a,b] = ?
                          [8.881781e-16,1.953126e-3] <enter>

Relative Fehlerschranke der gestörten Argumente ? 0 <enter>

Mantissenbreite der Teilintervalle (1e-5) ? 1e-5    <enter>
```

liefert das Programm `Poly1_relh.cpp` die relative Fehlerschranke

$$|\varepsilon_Q| \leq 1.111217 \cdot 10^{-16} = \varepsilon(Q), \quad \text{für alle Maschinenzahlen } \widetilde{x} \in A_2;$$

Damit sind wir jetzt in der Lage, mit Hilfe der Funktion `eps_4()` aus dem Modul `abs_relh` eine garantierte Oberschranke `S` für $|\varepsilon_{A_2}| \leq$ `S` zu berechnen. Mit dem folgenden Programm

```
#include <iostream>      // Wegen cout
#include "abs_relh.hpp"  // Wegen eps_3()
```

```
using namespace std;
using namespace cxsc;

int main(){
    const real eps_h = comp(0.5,-51);  // eps_h = 2^(-52)
    real S,eps_app,eps_Q,eps_sqrt(eps_h);
    cout << "eps(app) = ?" << endl;  cin >> RndUp >> eps_app;
    cout << "eps(Q) = ?"   << endl;  cin >> RndUp >> eps_Q;

    S = eps_4(eps_app,eps_Q,eps_sqrt,eps_h);
    cout << RndUp << "S = " << S << endl;
    return 0;
}
```

erhält man mit $\varepsilon(app) = 1.987371 \cdot 10^{-17}$ und $\varepsilon(Q) = 1.111217 \cdot 10^{-16}$ für $|\varepsilon_{A_2}|$ die folgende Oberschranke

$$|\varepsilon_{A_2}| \leq \mathtt{S} = 5.750847 \cdot 10^{-16} = \varepsilon(A_2)$$

8.6.3.3 Fehlerabschätzung in A_3

In $A_3 = [2^{-9}, 1)$ benutzen wir nach (8.85) die Darstellung

$$\operatorname{arcosh}(1+x) = \ln\left[1 + (x + \sqrt{2x+x^2})\right], \quad 2^{-9} \leq x < 1.0, \tag{8.91}$$

wobei $(x + \sqrt{2x+x^2})$ direkt ausgewertet und als Argument der **C-XSC** Funktion `lnp1()` verwendet wird. Der relative Fehler im Bereich A_3 ist damit der relative Auswertefehler der rechten Seite in (8.91); seine Berechnung erfolgt mit Hilfe des folgenden Programms

```
// Program for calculating the evaluating error in
// the range A_3 = [2^(-9),1]

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

using namespace cxsc;
using namespace std;
```

```
real T3(const interval& xi, const real& delx)
// T3(x) := ln[1 + (x+sqrt(2*x + x^2)) ]; A_3 = [2^(-9),1]
{
    real rel,delP,delX2,delA,delW;
    interval P,X2,A,W;
    abs_mulh1(xi,delx,interval(2),0.0,P,delP); // P: 2*xi
    abs_mulh1(xi,delx,xi,delx,X2,delX2); // X2: xi*xi
    abs_addh1(P,delP,X2,delX2,A,delA);  // A: 2*xi + xi*xi
    abs_sqrt_abs(A,delA,W,delW);  // W: sqrt(2*xi + xi*xi)
    abs_addh1(xi,delx,W,delW,A,delA); // A: xi + sqrt(2*xi + xi*xi)
    rel_lnp1_abs(A,delA,W,rel); // W: ln[1 + (x+sqrt(2*x + x^2)) ];

    return rel;
}

int main()
{
    interval X(comp(0.5,-8),comp(0.5,1));
    real bnd, diam = 1e-5, delx=0;

    Max_bnd_Xi(T3,X,delx,diam,bnd);
    cout << "Term T3(x): Rel. Schranke eps(f,X) = "
         << RndUp << bnd << endl;
}
```

Das obige Programm liefert die Bildschirmausgabe:

```
Term T3(x): Rel. Schranke eps(f,X) = 7.792706E-016
```

Der relative Auswertefehler ist damit gegeben durch: $\varepsilon(A_3) = 7.792706 \cdot 10^{-16}$

8.6.3.4 Fehlerabschätzung in A_4

In $A_4 = [1, 2^{+50})$ benutzen wir nach (8.86) die Darstellung

$$\operatorname{arcosh}(1+x) = \ln\left[1 + x \cdot (1 + \sqrt{1 + 2/x})\right], \quad 1 \le x < 2^{+50}, \tag{8.92}$$

wobei $x \cdot (1 + \sqrt{1 + 2/x})$ direkt ausgewertet und als Argument der **C-XSC** Funktion `lnp1()` verwendet wird. Der relative Fehler im Bereich A_4 ist damit der relative Auswertefehler der rechten Seite in (8.92); seine Berechnung erfolgt mit Hilfe des folgenden Programms

```
// Program for calculating the evaluating error in
// the range A_4 = [1,2^(+50)]

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

using namespace cxsc;
using namespace std;

real T4(const interval& xi, const real& delx)
// T4(x) := ln[1+ x*(1+sqrt(1+2/x))];  A_4 = [1,2^(+50)]
{
    real delq,delA,delW,delKl,rel;
    interval Q,A,W,Kl;
    abs_divh1(interval(2),0.0,xi,delx,Q,delq); // 2/xi
    abs_1px_delh(Q,delq,A,delA); // A: 1+2/xi
    abs_sqrt_abs(A,delA,W,delW); // W: sqrt(1+2/xi);
    abs_1px_delh(W,delW,Kl,delKl); // Kl: (1 + sqrt(1+2/xi));
    abs_mulh1(xi,delx,Kl,delKl,A,delA); // A: xi*(1+sqrt(1+2/xi));
    rel_lnp1_abs(A,delA,W,rel); // W: ln[1 + xi*(1+sqrt(1+2/xi))];

    return rel;
}


int main()
{
    interval X(comp(0.5,1),comp(0.5,51));
    real bnd, diam = 1e-5, delx=0;

    Max_bnd_Xi(T4,X,delx,diam,bnd);
    cout << "Term T4(x): Rel. Schranke eps(f,X) = "
         << RndUp << bnd << endl;
}
```

Das obige Programm liefert die Bildschirmausgabe:

```
Term T4(x): Rel. Schranke eps(f,X) = 5.368057E-016
```

Der relative Auswertefehler ist damit gegeben durch: $\varepsilon(A_4) = 5.368057 \cdot 10^{-16}$

8.6.3.5 Fehlerabschätzung in A_5

In $A_5 = [2^{50}, 2^{1022})$ benutzen wir die Approximation

$$\operatorname{arcosh}(1+x) \approx \ln(2x)\,, \quad 2^{50} \le x < 2^{1022}$$

Da das Argument $2x$ stets rundungsfehlerfrei berechnet wird, ist der relative Auswertefehler des Approximationsterms $\ln(2x)$ gleich der relativen Fehlerschranke der Logarithmusfunktion: $\varepsilon(\ln) = 2.9398 \cdot 10^{-16}$, vergl. die Tabelle C.1 auf Seite 415. Es muss zusätzlich der Approximationsfehler berücksichtigt werden. Nach (8.90) ist $\varepsilon_{app}(x) = 1/(x \cdot \operatorname{arcosh}(x))$ monoton fallend in x, so dass für eine Oberschranke des relativen Approximationsfehlers $\varepsilon_{app}(2^{50}) = 1/(2^{50} \cdot \operatorname{arcosh}(2^{50}))$ intervallmäßig auszuwerten ist. Man findet: $\varepsilon_{app}(2^{50}) < \varepsilon(app, A_5) = 2.512492 \cdot 10^{-17}$.
Nach (7.1) von Seite 186 wird der relative Gesamtfehler $\varepsilon(f)$ wie folgt berechnet:

$$\varepsilon(f) := \varepsilon(app, A_5) + \varepsilon(\ln) \cdot [1 + \varepsilon(app, A_5)]$$

Durch Intervallauswertung der rechten Seite findet man:

$$\varepsilon(f) < 3.191050 \cdot 10^{-16} = \varepsilon(A_5)$$

8.6.3.6 Fehlerabschätzung in A_6

In $A_6 = [2^{1022}, \texttt{MaxReal}]$ benutzen wir die Approximation

$$\operatorname{arcosh}(1+x) \approx \ln(2) + \ln(x)\,, \quad 2^{1022} \le x \le \texttt{MaxReal}$$

Wegen $\ln(2x) \equiv \ln(2) + \ln(x)$ vermeiden wir jetzt einen Overflow bei der Berechnung von $2x$. Um die aufwendige Auswertung von $\ln(2)$ zu umgehen, wird der zur nächsten Rasterzahl gerundete Wert

$$\texttt{c\_ln2\_B = 6243314768165359.0 / 9007199254740992.0} < \ln(2)$$

bei der Implementierung der Funktion $\operatorname{arcosh}(1+x)$ mit dem absoluten Höchstfehler $\Delta(\ln(2)) < 2.3190469 \cdot 10^{-17}$ gespeichert.

Mit dem folgenden Programm wird der relative Auswertefehler von $\texttt{c\_ln2\_B} \odot \widetilde{\ln}(x)$ abgeschätzt:

```
// Program for calculating the evaluating error in
// the range A_6 = [2^(+1022),MaxReal]

#include "abs_relh.hpp" // Wegen rel_addh1(),...
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <iostream>     // Wegen cout
```

```
using namespace cxsc;
using namespace std;

real T6(const interval& xi, const real& delx)
// T6(x) := c_ln2_B + ln(x);  A_6 = [2^(+1022),MaxReal]
{
    const interval ln_2 = ln(interval(2.0));
    interval LNx,IDEL,R;
    string("[2.3190469e-17,2.3190469e-17]") >> IDEL;
    real rel,delLNx,DEL(Sup(IDEL));

    abs_ln_abs(xi,delx,LNx,delLNx); // LNx: ln(xi)
    rel_addh1(ln_2,DEL,LNx,delLNx,R,rel); // R:
    return rel;
}

int main()
{
    interval X(comp(0.5,1023),MaxReal);
    real bnd, diam = 1e-5, delx=0;

    Max_bnd_Xi(T6,X,delx,diam,bnd);
    cout << "Term T6(x): Rel. Schranke eps(f,X) = "
         << RndUp << bnd << endl;
}
```

Das obige Programm liefert die Bildschirmausgabe:

```
Term T6(x): Rel. Schranke eps(f,X) = 5.157705E-016
```

Der relative Auswertefehler ist damit gegeben durch: $\varepsilon(f, A_6) = 5.157705 \cdot 10^{-16}$. Wie im Bereich A_5 ist jetzt noch der Approximationsfehler zu berücksichtigen. Nach (8.90) ist $\varepsilon_{app}(x) = 1/(x \cdot \operatorname{arcosh}(x))$ monoton fallend in x, so dass für eine Oberschranke des relativen Approximationsfehlers $\varepsilon_{app}(2^{1022}) = 1/(2^{1022} \cdot \operatorname{arcosh}(2^{1022}))$ intervallmäßig auszuwerten ist. Es gilt: $\varepsilon_{app}(2^{1022}) < \varepsilon(app, A_6) = 5.562685 \cdot 10^{-309}$. Dieser Approximationsfehler ist natürlich sehr klein, darf aber bei einer korrekten Fehlerabschätzung keinesfalls vernachlässigt werden! Nach (7.1) von Seite 186 wird der relative Gesamtfehler $\varepsilon(f)$ wie folgt berechnet:

$$\varepsilon(f) := \varepsilon(app, A_6) + \varepsilon(f, A_6) \cdot [1 + \varepsilon(app, A_6)]$$

Durch Intervallauswertung der rechten Seite findet man:

$$\varepsilon(f) < 5.157706 \cdot 10^{-16} = \varepsilon(A_6)$$

Damit haben wir die Oberschranken der relativen Fehler in allen sechs Teilbereichen berechnet, und im Gesamtbereich $0 \le x \le$ `MaxReal` ist der relative Fehler das Maximum der Oberschranken in den sechs Teilbereichen:

$$\left|\frac{\operatorname{arcosh}(1+\widetilde{x}) - g_i(\widetilde{x})}{\operatorname{arcosh}(1+\widetilde{x})}\right| \le 7.792706 \cdot 10^{-16} =: \varepsilon(\text{acoshp1}) \quad \forall \widetilde{x} \in [0, \texttt{MaxReal}]$$

Anmerkungen:

1. Da außer dem exakt berechneten Funktionswert $\operatorname{arcosh}(1+0) = 0$ alle anderen Funktionswerte im normalisierten Bereich liegen, ist die Berechnung eine absoluten Fehlerschranke nicht erforderlich.

2. Im Bereich $A_3 = [2^{-9}, 1)$ wird die relative Fehlerschranke $\varepsilon(A_3)$ maximal. Der benutzte Term $\ln[1 + (x + \sqrt{2x + x^2})]$ scheint dennoch optimal gewählt zu sein, da alle möglichen Umformungen des Arguments $(x + \sqrt{2x + x^2})$ stets noch größere Oberschranken als $\varepsilon(A_3) = 7.792706 \cdot 10^{-16}$ lieferten.

8.6.4 Intervallargumente

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

$$W_{\texttt{x}} := \left\{ y \in I\!R \;\middle|\; y = \operatorname{arcosh}(1+x) \;\wedge\; x \in \texttt{x} \right\} \subset \texttt{W} \in IR \tag{8.93}$$

gesucht. Da $f(x) = \operatorname{arcosh}(1+x)$ für alle $x \ge 0$ monoton wächst, ist die Berechnung der gesuchten Einschließung `W` recht einfach:

Zur Berechnung von `Inf(W)` bestimmt man zunächst `r1=acoshp1(Inf(x))` und muss dann diesen Maschinenwert noch mit `q_acoshp1m` < 1 multiplizieren, um eine garantierte Unterschranke aller Funktionswerte $f(x)$, mit $x \in$ `x` zu erhalten. Mit Hilfe der obigen Fehlerschranke $\varepsilon(f)$ wird `q_acoshp1m` vorher durch Aufruf der Funktion `eps2fractions(...)` aus dem Modul `bnd_util` berechnet. Weitere Einzelheiten dazu findet man auf Seite 81.

Die Berechnung von `Sup(W)` erfolgt ganz entsprechend. Man berechnet zunächst wieder `r2=acoshp1(Sup(x))` und muss dann noch abschließend mit der Konstanten `q_acoshp1p` > 1 multiplizieren, um eine garantierte Oberschranke aller Funktionswerte $f(x)$, mit $x \in$ `x` zu erhalten.

8.6.5 Numerische Ergebnisse

Intervallargumente:

1. $\mathtt{x} = [0, 0]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,0.000000000000000E+000]
2. $\mathtt{x} = [\mathtt{minreal}, \mathtt{minreal}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.143455569405253E-162,3.143455569405263E-162]
3. $\mathtt{x} = [0, \mathtt{minreal}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,3.143455569405263E-162]
4. $\mathtt{x} = [\mathtt{MinReal}, \mathtt{MinReal}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [2.109537322972597E-154,2.109537322972604E-154]
5. $\mathtt{x} = [\mathtt{minreal}, \mathtt{MinReal}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.143455569405253E-162,2.109537322972604E-154]
6. $\mathtt{x} = [2^{-200}, 2^{-200}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.115617790989470E-030,1.115617790989474E-030]
7. $\mathtt{x} = [2^{-9}, 2^{-9}]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [6.248983194170978E-002,6.248983194170996E-002]
8. $\mathtt{x} = [1, 1]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.316957896924815E+000,1.316957896924819E+000]
9. $\mathtt{x} = [2^{-200}, 1]$
   $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.115617790989470E-030,1.316957896924819E+000]
10. $\mathtt{x} = [1025, 1025]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [7.626569968800589E+000,7.626569968800610E+000]
11. $\mathtt{x} = [2^{50}, 2^{50}]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [3.535050620855717E+001,3.535050620855728E+001]
12. $\mathtt{x} = [2^{1022}, 2^{1022}]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [7.090895657128230E+002,7.090895657128252E+002]
13. $\mathtt{x} = [\mathtt{MaxReal}, \mathtt{MaxReal}]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [7.104758600739429E+002,7.104758600739451E+002]
14. $\mathtt{x} = [0, \mathtt{MaxReal}]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [0.000000000000000E+000,7.104758600739451E+002]
15. $\mathtt{x} = [1.33, 1.33]$
    $\leadsto \quad W_{\mathtt{x}} \subset \mathtt{W} \subseteq$ [1.489413784903228E+000,1.489413784903233E+000]

8.6.6 Quelltext für Punkt- und Intervallargumente

```
// Program for evaluation of the function acoshp1 = acosh(1+x)
// for point and interval arguments x in [0,+MaxReal]

#include <imath.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

static real q_acoshp1[5] = // Polynomial coefficients of Q_4(x)
          // roundet to nearest. acosh(1+x) = sqrt(2*x)*Q_4(x)
{ 1.0 / 1.0,                                    // q_acoshp1[0]
  -6004799503160661.0 / 72057594037927936.0,
  +5404319552844595.0 / 288230376151711744.0,
  -6433713753386423.0 / 1152921504606846976.0,
  +8756999275442631.0 / 4611686018427387904.0   // q_acoshp1[4]
};

static const real c_ln2_B = 6243314768165359.0 / 9007199254740992.0;
// c_ln2_B < ln(2) is the nearest machine number for ln(2) with an
// absolute error < 2.3190469E-17;

real acoshp1(const real& x) throw()
// acoshp1(x) = acosh(1+x);  rel. error: eps = 7.792706E-16 = e(f)
// Ausfuehrlich getestet;  Blomquist, 27.03.05;
{
    real res;
    int ex(expo(x));
    if (x<0)
        cxscthrow(STD_FKT_OUT_OF_DEF("real acoshp1(const real&)"));
    // For argument x now it holds: 0 <= x <= MaxReal;
    if (ex<=-50) res = sqrt(2*x); // 0<=x<2^(-50): acoshp1(x)=sqrt(2x)
    else if (ex<=-9) // 2^(-50)<=x<2^{-9}: acoshp1(x)=sqrt(2x)*Q_4(x)
      res = sqrt(2*x)*((((q_acoshp1[4]*x+q_acoshp1[3])*x+q_acoshp1[2])
                          *x+q_acoshp1[1])*x + q_acoshp1[0]);
    else if (ex<=0) res = lnp1(x+sqrt(2*x+x*x));     // range A_3
    else if (ex<=50) res = lnp1(x*(1+sqrt(1+2/x)));  // range A_4
    else if (ex<=1022) res = ln(2*x);                // range A_5
    else res = ln(x) + c_ln2_B;                      // range A_6
```

```
    return res;
} // acoshp1

// Constants for the interval function acoshp1(x) = acosh(1+x):
// (1+e(f)):
static const real q_acoshp1p(4503599627370503.0/4503599627370496.0);
// (1-e(f))
static const real q_acoshp1m(9007199254740981.0/9007199254740992.0);

interval acoshp1(const interval& x)
// acoshp1;  Blomquist, 28.03.2005;
{
    real r1,r2,sx,ix;
    sx = Sup(x);  ix = Inf(x);
    // Calculating of the lower bound r1:
    r2 = acoshp1(ix);
    r1 = r2 * q_acoshp1m;
    // Calculating of the upper bound r2:
    r2 = (sx==ix)? r2*q_acoshp1p : acoshp1(sx)*q_acoshp1p;
    return interval(r1,r2);

} // acoshp1 (nterval arguments)

int main()
{
    interval z,w;

    while (1) {
        cout << "interval z = [a,b] = ?" << endl;   cin >> z;
//      z = interval(comp(0.5,-8),comp(0.5,-8));
        w = acoshp1(z);
        cout << SetPrecision(15,15)
             << Scientific << "acoshp1(z) = " << w << endl << endl;
    }; // end of while
}
```

Kapitel 9

Staggered Arithmetik

Die Grundlagen einer staggered Arithmetik findet man in [20],[26],[28]. In **C++**basiert die staggered Arithmetik auf den Typen *real* und *interval*. In den Klassen `l_real` und `l_interval` sind alle dazu notwendigen Operatoren und Methoden definiert und können mit Hilfe der Dateien `l_real.hpp` und `l_interval.hpp` in entsprechende **C++** Programme eingebunden werden. Vergleichen Sie dazu bitte die Programme auf Seite 360 und 364.

9.1 Implementierung in C++

9.1.1 Variablen vom Typ *l_real*

In **C++** wird eine Variable x vom Typ *l_real* intern gespeichert als ein Vektor mit n Komponenten x_i vom Typ *real*:

$$x = \sum_{i=1}^{n} x_i, \quad x \in \mathbb{R}, \quad x_i \in S(2,53), \quad n = \texttt{StagPrec(x)} \geq 1, \tag{9.1}$$

wobei die jeweilige Präzision der Variablen x mit Hilfe der Funktion `StagPrec(x)` bestimmt werden kann und nicht mit der durch `stagprec` ≥ 1 vorgegebenen Präzision der eigentlichen staggered Rechnungen zu verwechseln ist. Die im Modul `l_real` deklarierte Variable `stagprec` wird dort mit `stagprec` $= 2$ initialisiert, kann vom Anwender in einem Programm jedoch dynamisch verändert werden und bestimmt die Präzision der staggered Arithmetik.

In einem ersten Programmbeispiel wird gezeigt, dass bei der Auswertung des Terms $(2^{511} + 2^{-537}) \cdot (2^{511} - 2^{-537}) = 2^{1022} - 2^{-1074}$ das vollständige Ergebnis $2^{1022} - 2^{-1074}$ schon mit $n = 2$ exakt dargestellt werden kann. Die **C-XSC** Funktion `int StagPrec(l_real& x)` liefert die momentane Präzision der staggered Variablen `x`, und der Operator `[ ]` ermöglicht den Zugriff auf die entsprechenden Komponenten

von x. Dieser Operator [ ] sollte jedoch aus Sicherheitsgründen nur zur Anwendung kommen, wenn die Präzision von x genau bekannt ist!

```
// Programm staggered1.cpp
#include <l_real.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

int main()
{
    l_real x,y,z;
    real r1,r2;
    int n;
    r1 = comp(0.5,+512);  // r1 = 2^(+511)
    r2 = comp(0.5,-536);  // r2 = 2^(-537)
    x = r1;
    z = (x+r2)*(x-r2);    // z = 2^(+1022) - 2^(-1074)
    n = StagPrec(z);
    cout << "Praezision n von z:   n = " << n << endl;
    if (z[1]==comp(0.5,1023)) cout << "z[1] = +2^(+1022)" << endl;
    if (z[2]==-minreal)       cout << "z[2] = -2^(-1074)" << endl;
    stagprec = 20;
    x = 1;
    z = x / 3;
    cout << Scientific << SetDotPrecision(16*21,16*21)
         << "1/3 = " << z << endl;
}
```

Das Programm staggered1.cpp liefert die folgende Ausgabe:

```
Praezision n von z:   n = 2
z[1] = +2^(+1022)
z[2] = -2^(-1074)
1/3 = 3.333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333316864478471958E-0001
```

Das erste Beispiel zeigt, dass schon mit den beiden *real*-Zahlen z[1] und z[2] das Ergebnis $2^{+1022} - 2^{-1074}$ exakt gespeichert werden kann. Im zweiten Beispiel kann

der Wert 1/3 weder im binären noch im dezimalen Zahlensystem exakt dargestellt werden. Wegen `minreal` $= 2^{-1074} = 4.9\ldots\cdot 10^{-324}$ lässt sich daher 1/3 mit höchstens 324 korrekten dezimalen Stellen angegeben, selbst wenn man `stagprec` $= 20$ noch weiter vergrößert. Daraus ergibt sich die folgende Grundregel:

> Wird ein nicht darstellbares Ergebnis einer staggered-correction Rechnung betragsmäßig zu klein, so kann unabhängig vom gewählten `stagprec`-Wert das Ergebnis nur noch mit einer stark begrenzten Stellenzahl berechnet werden.

Das folgende Programm `staggered2.cpp` demonstriert, dass die durch `stagprec` vorgegebene Präzision einer staggered Rechnung und die mit Hilfe der Funktion `StagPrec(x)` bestimmte Präzision der Variablen `x` durchaus verschieden sein können. Das Programm zeigt außerdem, dass die Darstellung von `x` durch die Summe in (9.1) nicht eindeutig ist.

```
// Programm staggered2.cpp
#include <l_real.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

int main()
{
    l_real x;
    real r = 6;
    cout << "Vordefinierte Praezision stagprec = " << stagprec <<endl;
    stagprec = 3;
    cout << "Momentane Praezision     stagprec = " << stagprec <<endl;
    cout << "Praezision von x = " << StagPrec(x) << endl;
    x = 6;
    cout << "Praezision von x = " << StagPrec(x) << endl;
    x = x + 0;
    cout << "Praezision von x = " << StagPrec(x) << endl;
```

```
    x[1] = 1;
    x[2] = -1;
    x[3] = 0;
    cout << "x[1] = " << x[1] << endl;
    cout << "x[2] = " << x[2] << endl;
    cout << "x[3] = " << x[3] << endl << endl;
    x = x + 0;
    cout << "x[1] = " << x[1] << endl;
    cout << "x[2] = " << x[2] << endl;
    cout << "x[3] = " << x[3] << endl;
    stagprec = 2;
    cout << "Neue Praezision:  stagprec = " << stagprec << endl;
    cout << "Alte Praezision von x = " << StagPrec(x) << endl;
    x = x + 1;
    cout << "Neue Praezision von x = " << StagPrec(x) << endl;
}
```

Das obige Programm `staggered2.cpp` liefert die folgende Ausgabe:

```
Vordefinierte Praezision stagprec = 2
Momentane Praezision     stagprec = 3
Praezision von x = 2
Praezision von x = 1
Praezision von x = 3
x[1] =   1.000000
x[2] =  -1.000000
x[3] =   0.000000

x[1] =   0.000000
x[2] =   0.000000
x[3] =   0.000000
Neue Praezision:  stagprec = 2
Alte Praezision von x = 3
Neue Praezision von x = 2
```

Beachten Sie bitte, dass gegen Ende des obigen Programms `staggered2.cpp` durch `stagprec` = 2 die Präzision der nachfolgenden staggered Arithmetik auf 2 gesetzt wird. Unmittelbar danach behält jedoch die Variable `x` noch ihre alte Präzision 3. Erst mit der Anweisung `x = x + 1;` erhält dann `x` die Präzision 2, da diese Anweisung eine staggered Addition mit der neuen Präzision 2 realisiert.

9.1.2 Variablen vom Typ *l_interval*

In **C++** wird eine Variable $\boldsymbol{x}$ vom Typ *l_interval* intern gespeichert als ein Vektor mit $n+1$ Komponenten x_i vom Typ *real*:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_{n+1}), \quad x_i \in S(2,53), \quad n = \texttt{StagPrec(x)} \geq 1, \tag{9.2}$$

Mathematisch ist obige Darstellung zu interpretieren als:

$$\boldsymbol{x} = \sum_{i=1}^{n-1} x_i + \boldsymbol{z}, \quad \boldsymbol{z} = [x_n, x_{n+1}], \quad x_j \in S(2,53), \quad j = 1,2,\ldots,n+1; \tag{9.3}$$

Für $n = 1$ gilt damit $\boldsymbol{x} = \boldsymbol{z}$, und für $n \geq 2$ ergibt sich:

$$\texttt{Inf}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i, \qquad \texttt{Sup}(\boldsymbol{x}) = \sum_{i=1}^{n-1} x_i + x_{n+1};$$

Beachten Sie bitte: Eine Variable `x` vom Typ *l_real* mit der Präzision n besitzt genau n *real*-Komponenten, und eine Variable `x` vom Typ *l_interval* mit der gleichen Präzision n besitzt $n+1$ *real*-Komponenten.

Das nachfolgende Programm `staggered3.cpp` liefert einfache Anwendungen zu den Datentypen *l_real* und *l_interval*. Die beiden Funktionen

```
l_real Inf(l_interval& x)   und   l_real Sup(l_interval& x)
```

liefern das Infimum bzw. Supremum von `x`, wobei der Rückgabewert vom Typ *l_real* unabhängig vom aktuellen `stagprec`-Wert die gleiche Präzision erhält wie die Variable `x` selbst. Damit wird erreicht, dass Infimum bzw. Supremum ohne Genauigkeitsverluste berechnet werden.

Die Funktion

```
l_interval adjust(l_interval& x)
```

liefert ein Rückgabeintervall vom Typ *l_interval*, das `x` bezüglich der durch `stagprec` vorgegebenen Präzision optimal einschließt; vergleichen Sie dazu die Ausgaben der Variablen `t` und `x` auf Seite 365.

Beachten Sie bitte, dass auch jetzt die Darstellung (9.2) nicht eindeutig ist, da z.B. für $n = 3$ die folgende Identität gilt:

$$\boldsymbol{x} = (5,0,1,4) \equiv (6,0,0,3) \equiv (0,0,6,9)$$

Das obige Beispiel zeigt außerdem, dass $\boldsymbol{x}$ nicht das Null-Intervall sein muss, nur weil die ersten Komponenten verschwinden. Dagegen ist $\boldsymbol{x}$ stets ein Punktintervall, wenn die beiden letzten Komponenten übereinstimmen, d.h. wenn gilt: $x_n = x_{n+1}$.

Das folgende Programm `staggered3.cpp`

```
// Programm staggered3.cpp
#include <l_interval.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

int main()
{
    l_real r1,r2;
    int p;
    l_interval x,y,t;
    cout << "Vorgegebene Praezision stagprec = " << stagprec << endl;
    stagprec = 3;
    cout << "Momentane Praezision     stagprec = " << stagprec <<endl;
    cout << "Praezision von x  : " << StagPrec(x) << endl;
    x = l_interval(1)/3;  p = StagPrec(x);
    cout << "Praezision von x  : " << p << endl;
    cout << Scientific << SetDotPrecision(16*p,16*p)
         << "x = " << x << endl;
    y = x;
    cout << "Praezision von y  : " << StagPrec(y) << endl;
    cout << "Praezision von r1 : " << StagPrec(r1) << endl;
    stagprec = 2;
    cout << "Jetzige Praezision stagprec = " << stagprec <<endl;
    r1 = Inf(x);  r2 = Sup(x);
    cout << "Praezision von r1 ist dennoch " << StagPrec(r1) << endl;
    if ((r1[3]!=r2[3]) && (x[3]!=x[4]))
        cout << "x ist kein Punktintervall" << endl;
    if ((r1[3]==x[3]) && (r2[3]==x[4]))
        cout << "Es gilt:   r1[3]=x[3]  und  r2[3]=x[4]." << endl;
    t = adjust(x); // t ist Einschliessung von x.
    cout << "Praezision von t  : " << StagPrec(t) << endl;
    cout << Scientific << SetDotPrecision(16*stagprec,16*stagprec)
         << "t = " << t << endl;
    if (x<t && x==y) cout << "t ist Einschliessung von x = y."<< endl;
}
```

liefert die folgende Ausgabe:

```
Vorgegebene Praezision stagprec = 2
Momentane Praezision     stagprec = 3
Praezision von x  : 2
Praezision von x  : 3
x = [3.33333333333333333333333333333333333333333333331E-0001,
     3.33333333333333333333333333333333333333333333338E-0001]
Praezision von y  : 3
Praezision von r1 : 2
Jetzige Praezision stagprec = 2
Praezision von r1 ist dennoch 3
x ist kein Punktintervall
Es gilt:   r1[3]=x[3]  und  r2[3]=x[4].
Praezision von t  : 2
t = [3.33333333333333333333333333333329E-0001,
     3.33333333333333333333333333333336E-0001]
t ist Einschliessung von x = y.
```

Beachten Sie auch jetzt, dass der Operator `[ ]` nur dann zur Anwendung kommt, wenn die Präzision der benutzten Variablen vorher genau bekannt ist. Andernfalls greift man auf nicht definierte Speicherbereiche zu und riskiert dabei völlig falsche Ergebnisse oder einen Programmabbruch!

9.2 Anmerkungen zur Fehlerabschätzung

Ziel dieses Abschnittes ist es, diejenigen Grundideen aufzuzeigen, mit denen die Fehlerabschätzungen für Standardfunktionen in eine Staggered-Correction Arithmetik durchzuführen sind. Mit den so berechneten Fehlerschranken sollen wir dann in der Lage sein, zu einem vorgegebenen Punkt- oder Intervallargument $\boldsymbol{x}$ eine garantierte und möglichst enge Einschließung aller Funktionswerte $f(x)$, mit $x \in \boldsymbol{x}$ zu berechnen. Um dabei eine Genauigkeit zu erhalten, die mit der durch `stagprec` vorgegebenen Präzision vergleichbar ist, müssen wir die folgende Grundforderung stellen:

> Damit die Genauigkeit einer Staggered-Correction Rechnung mit der durch `stagprec` vorgegebenen Präzision vergleichbar wird, darf der relative Durchmesser des Argumentintervalls $\boldsymbol{x}$ vom Typ *l_interval* den Wert $10^{-16 \cdot \texttt{stagprec}}$ nicht wesentlich übersteigen.

Wir verdeutlichen den Inhalt dieser Grundforderung am einfachsten mit Hilfe des folgenden Programms `staggered4.cpp`:

```
// Programm staggered4.cpp
// Zur Demonstration des Einflusses des relativen Durchmessers eines
// Argumentintervalls x auf die Genauigkeit einer staggered Rechnung.

#include <l_interval.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

int main()
{
    l_interval x,y;
    stagprec = 3;
    cout << "Gewaehlte Praezision:  stagprec = " << stagprec << endl;
    x = 3;  // x ist Punktintervall
    cout << "Praezision von x:  " << StagPrec(x) << endl;
    cout << SetDotPrecision(16,16)
         << "Rel. Durchmesser von x: " << diam(x)/Inf(x) << endl;
    y = 1/x;
    cout << Scientific << SetDotPrecision(16*stagprec,16*stagprec+1)
         << "1/x = " << y << endl;

    x = x + interval(0,1e-48);
    cout << "Praezision von x:  " << StagPrec(x) << endl;
    cout << SetDotPrecision(16,16)
         << "Rel. Durchmesser von x: " << diam(x)/Inf(x) << endl;
    y = 1/x;
    cout << Scientific << SetDotPrecision(16*stagprec,16*stagprec+1)
         << "1/x = " << y << endl;

    x = 3;
    x = x + interval(0,1e-32);
    cout << "Praezision von x:  " << StagPrec(x) << endl;
    cout << SetDotPrecision(16,16)
         << "Rel. Durchmesser von x: " << diam(x)/Inf(x) << endl;
    y = 1/x;
    cout << Scientific << SetDotPrecision(16*stagprec,16*stagprec+1)
         << "1/x = " << y << endl;
}
```

Das umseitige Programm `staggered4.cpp` liefert die folgende Ausgabe:

```
Gewaehlte Praezision:  stagprec = 3
Praezision von x:  1
Rel. Durchmesser von x: 0.0000000000000000
1/x = [3.3333333333333333333333333333333333333333333333310E-0001,
       3.3333333333333333333333333333333333333333333333379E-0001]
Praezision von x:  3
Rel. Durchmesser von x: 3.333333333333332E-0049
1/x = [3.3333333333333333333333333333333333333333333333173E-0001,
       3.3333333333333333333333333333333333333333333333379E-0001]
Praezision von x:  3
Rel. Durchmesser von x: 3.333333333333335E-0033
1/x = [3.3333333333333333333333333333332222222222222222143E-0001,
       3.3333333333333333333333333333333333333333333333379E-0001]
```

Man erkennt deutlich, dass in den ersten beiden Beispielen mit den relativen Durchmessern 0.0 und $3.33\ldots \cdot 10^{-49}$ die Genauigkeit mit der durch `stagprec` $= 3$ vorgegebenen Präzision übereinstimmt, da jeweils etwa $16 \cdot$ `stagprec` $= 48$ korrekte dezimale Ziffern berechnet werden. Im letzten Beispiel erhält man jedoch nur noch 32 korrekte dezimale Ziffern, da jetzt der relative Durchmesser $3.33\ldots \cdot 10^{-33}$ deutlich größer ist als $10^{-16 \cdot \texttt{stagprec}} = 10^{-48}$. Beachten Sie bitte, dass im letzten Beispiel die Genauigkeit auch nicht dadurch vergrößert werden kann, dass der Wert von `stagprec` weiter erhöht wird!

Die auf Seite 361 für den Datentyp *l_real* formulierte Grundregel lässt sich wörtlich auch auf den Datentyp *l_interval* übertragen:

> Wird ein nicht darstellbares Ergebnis einer staggered-correction Rechnung betragsmäßig zu klein, so kann unabhängig vom gewählten `stagprec`-Wert das Ergebnis nur noch mit einer stark begrenzten Stellenzahl berechnet werden.

Diese zu starken Ungenauigkeiten könne jedoch oft durch eine geeignete Skalierung zu kleiner Zwischenergebnisse mit einer Potenz 2^m vermieden werden, vgl. Sie dazu bitte das Beispiel $f(x) = \sqrt{1 - x^2}$ auf Seite 380.

Für Standardfunktionen vom Typ *real* werden absolute oder relative Fehlerschranken für alle Maschinenargumente des jeweiligen Definitionsbereichs angegeben, um damit zu einem vorgegeben Argumentintervall $\boldsymbol{x}$ eine garantierte Einschlie-

ßung aller Funktionswerte $f(x)$ für $x \in \boldsymbol{x}$ berechnen zu können. Bei monotonen Funktionen werden Infimum und Supremum dieser Einschließungen berechnet, indem an den Randpunkten von $\boldsymbol{x}$ die Standardfunktionen einzeln ausgewertet werden und mit Hilfe der bekannten Fehlerschranken geeignet auf- bzw. abgerundet wird. Nach diesem Verfahren können die Einschließungen auch für breite Argumentintervalle $\boldsymbol{x}$ nahezu optimal berechnet werden. Da wir jedoch bei einer Staggered-Correction Arithmetik für eine hohe Genauigkeit auf nahezu punktförmige Argumentintervalle $\boldsymbol{x}$ angewiesen sind, gehen wir bei der Realisierung der staggered Standardfunktionen einen anderen Weg. Wird $f(x)$ z.B. durch eine Taylorreihe $T_N(x)$ approximiert

$$f(x) \approx T_N(x),$$

so wird $T_N(x)$ intervallmäßig ausgewertet und damit eine Abschätzung des Auswertefehlers überflüssig, da diese intervallmäßige Auswertung automatisch eine garantierte Einschließung von $T_N(x)$ für alle $x \in \boldsymbol{x}$ liefert. Es muss daher nur noch eine Schranke des absoluten Approximationsfehlers angegeben werden, wobei der Polynomgrad N so zu wählen ist, dass die durch `stagprec` vorgegebene Genauigkeit erreicht werden kann. Im folgenden Abschnitt werden diese Schritte am Beispiel der hyperbolischen Cotangens-Funktion ausführlich beschrieben.

9.3 coth(x)

Zu einem vorgegebenen und nicht zu breiten Argumentintervall $\boldsymbol{x}$ vom Typ *l_interval* ist gesucht eine garantierte und möglichst enge Einschließung der Funktionswerte $f(x) = \coth(x)$ für alle reellen $x \in \boldsymbol{x}$, mit $0 \notin \boldsymbol{x}$.

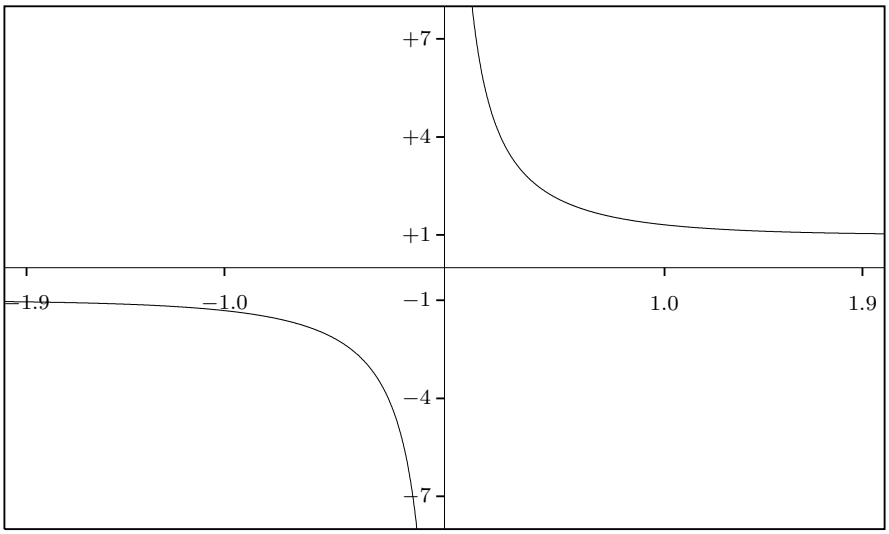


Abbildung 9.1: coth-Funktion

Die coth-Funktion ist definiert durch:

$$\coth(x) = \frac{\cosh(x)}{\sinh(x)} = \frac{e^x + e^{-x}}{e^x - e^{-x}} = 1 + \frac{2}{e^{2x} - 1} \tag{9.4}$$

Im Ursprung besitzt $\coth(x)$ eine Polstelle mit Vorzeichenwechsel, und für $0 < |x| < \pi$ gilt nach [1] die folgende Laurent-Entwicklung:

$$\coth(x) = \frac{1}{x} + \frac{x}{3} - \frac{x^3}{45} + \frac{2}{945}x^5 - + \ldots + \frac{2^{2n}B_{2n}}{(2n)!}x^{2n-1}, \tag{9.5}$$

wobei B_n die Bernoullischen Zahlen sind: $B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30}, \ldots$ Weitere Informationen findet man in [1, Seite 804].

Für die spätere Fehlerabschätzung ist es wichtig zu wissen, dass für $x > 0$ die Reihe in (9.5) eine alternierende Leibniz-Reihe ist. Bricht man diese Reihe ab, so ist der Fehler höchstens so groß wie der Betrag des ersten weggelassenen Summanden, [17, Seite 63], vgl. auch Seite 188.

9.3.1 Der Algorithmus

Das vorgegebene Argumentintervall $\boldsymbol{x}$ vom Typ *l_interval* wird zunächst optimal in das Intervall `dx` vom Typ *interval* gerundet, so dass gilt $\boldsymbol{x} \subseteq$ `dx`. Die Programmanweisung `einfachgenau = coth(dx);` liefert dann mit `einfachgenau` vom Typ *interval* eine garantierte Einschließung aller Funktionswerte $f(x) = \coth(x)$, mit $x \in \boldsymbol{x}$. Bezeichnen wir mit `y` vom Typ *l_interval* eine Einschließung der Funktionswerte $f(x) = \coth(x)$, wobei `y` in der Staggered-Correction Arithmetik berechnet sein soll, so kann z.B. bei einem zu breiten Intervallargument $\boldsymbol{x}$ wegen der unvermeidbaren Intervallüberschätzungen dieses `y` mit einem größeren Intervalldurchmesser berechnet worden sein als die Einschließung `einfachgenau`. In einem solchen Fall liefert dann der Durchschnitt `y` $\cap$ `einfachgenau` eine optimale Einschließung der Funktionswerte $f(x) = \coth(x)$, mit $x \in \boldsymbol{x}$. In der Sprachumgebung **C-XSC** wird der Durchschnitt mit Hilfe des Operators `&` realisiert, (`using namespace cxsc`).

Im Falle $0 \in \boldsymbol{x}$ gilt auch $0 \in$ `dx`, und bei der Berechnung von cot(`dz`) erfolgt eine entsprechende Fehlermeldung mit Programmabbruch.

Wegen der Punktsymmetrie $f(-x) \equiv -f(x)$ kann man sich auf Intervallargumente $\boldsymbol{x}$ mit `Inf`$(\boldsymbol{x}) > 0$ beschränken. Der Bereich [`MinReal`, `MaxReal`] wird in vier Teilbereiche A_i unterteilt[1]:

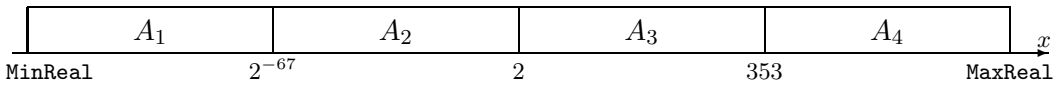


Abbildung 9.2: Teilintervalle A_i des Bereichs [`MinReal`, `MaxReal`]

Zu vorgegebenem Argumentintervall $\boldsymbol{x}$ wird eine Einschließung von $\coth(\boldsymbol{x})$ berechnet durch:

$$\coth(\boldsymbol{x}) \subseteq \begin{cases} \dfrac{1}{\boldsymbol{x}} + \dfrac{\boldsymbol{x}}{3} - \dfrac{\boldsymbol{x}^3}{45} + - \ldots & \text{falls } \texttt{Inf(dx)} < 2^{-67} \\ \dfrac{\cosh(\boldsymbol{x})}{\sinh(\boldsymbol{x})} & \text{falls } \texttt{Inf(dx)} < 2 \\ 1 + \dfrac{2}{e^{2\boldsymbol{x}} - 1} & \text{falls } \texttt{Inf(dx)} < 353 \\ [1, 1 + 2^{-1017}] & \text{falls } \texttt{Inf(dx)} \geq 353 \end{cases} \tag{9.6}$$

In A_2, A_3 erhält man damit automatisch garantierte Einschließungen, da identische Funktionsterme intervallmäßig ausgewertet werden. In den Bereichen A_1, A_4 liefern die entsprechenden Fehlerabschätzungen ebenfalls Einschließungen für $\coth(\boldsymbol{x})$, und zwar auch dann, wenn $\boldsymbol{x}$ einen beliebigen Durchmesser besitzt.

[1] Wählt man kleinere Argumente als `MinReal`, so liefert die staggered Auswertung von 1/x in der Laurent-Reihe eine Fehlermeldung mit Programmabbruch.

9.3.2 Fehlerabschätzung in A_1

Ist $\boldsymbol{x}$ ein vorgegebenes staggered Intervall vom Typ *l_interval* und gilt $\texttt{Inf}(\boldsymbol{x}) < 2^{-67}$, so wird $\coth(\boldsymbol{x})$ zunächst approximiert durch:

$$\coth(\boldsymbol{x}) \approx \frac{1}{\boldsymbol{x}} + \frac{\boldsymbol{x}}{3} \cdot P_N(u), \quad P_N(u) := \sum_{j=0}^{N} c_j\, u^j,\ u = x^2,\ 1 \le N \le 7; \tag{9.7}$$

Die acht Polynomkoeffizienten c_j sind definiert durch:

$$c_0 = 1,\ \ c_1 = \frac{-1}{15},\ \ c_2 = \frac{+2}{315},\ \ c_3 = \frac{-1}{1575},\ \ c_4 = \frac{+2}{31185},\ \ c_5 = \frac{-1382}{212837625},$$
$$c_6 = \frac{+4}{6081075},\ \ c_7 = \frac{-3617}{54273594375},$$

wobei Zähler und Nenner aller c_j rundungsfehlerfrei im *double*-Format gespeichert werden können. Mit den in (9.5) angegebenen Koeffizienten $a_n := (2^{2n} \cdot B_{2n})/(2n)!$ besteht der Zusammenhang

$$a_n := \frac{2^{2n} \cdot B_{2n}}{(2n)!} = \frac{1}{3} \cdot c_{n-1}, \quad n = 1, 2, \ldots \tag{9.8}$$

Wir zeigen zunächst, dass die Reihe in (9.5) und damit auch das Polynom $P_N(u)$ für $0 \le x < \pi$ eine Leibniz-Reihe ist, wobei das notwendige Alternieren dieser Reihe durch die alternierenden Bernoulli-Zahlen gesichert ist. Da die Reihe in (9.5) für $0 \le x < \pi$ konvergiert, bilden die Reihenglieder eine Nullfolge, und es bleibt dann nur noch zu zeigen, dass die Beträge der Reihenglieder monoton fallen, d.h. mit

$$b_n(x) := \frac{4^n \cdot |B_{2n}| \cdot x^{2n-1}}{(2n)!}, \quad n = 1, 2, \ldots, \quad 0 < x < 2^{-67} \tag{9.9}$$

ist für alle $n \in \mathbb{N}$ zu zeigen: $b_{n+1}(x)/b_n(x) < 1$. Zum Beweis benutzen wir die nach [1, Seite 805] gültige Ungleichung

$$\frac{2(2n)!}{(2\pi)^{2n}} \left(\frac{1}{1 - 2^{1-2n}} \right) > |B_{2n}| > \frac{2(2n)!}{(2\pi)^{2n}}, \qquad n = 1, 2, \ldots$$

Nach einfachen Umrechnungen erhält man daraus

$$\frac{b_{n+1}(x)}{b_n(x)} < \left(\frac{x}{\pi}\right)^2 \cdot \frac{1}{1 - 2^{-2n-1}} \le \left(\frac{x}{\pi}\right)^2 \cdot \frac{8}{7}$$

und unter der Voraussetzung $x < \pi\sqrt{7/8}$, die für $0 < x < 2^{-67}$ sicher erfüllt ist, gilt daher für alle $n \ge 1$ die behauptete Ungleichung: $b_{n+1}(x)/b_n(x) < 1$ ∎

Zu einem vorgegebenen Argumentintervall $\boldsymbol{x}$ vom Typ *l_interval* können wir damit einen beliebigen Polynomgrad N wählen und die rechte Seite von (9.7) intervallmäßig auswerten. Die berechnete Einschließung bezeichnen wir mit `y`. Der Approximationsfehler ist dann nach (9.8) und (9.9) gegeben durch $b_{N+1}(x)$ und sein Maximum wird für $x = \texttt{Sup(dx)}$ berechnet:

$$b_{N+1}(x) \leq b_{N+1}(\texttt{Sup(dx)}) =: \alpha$$

Die gesuchte Einschließung für $\coth(\boldsymbol{x})$ ist dann gegeben durch[2]:

$$\coth(\boldsymbol{x}) \subseteq \texttt{y} + [-\alpha, +\alpha] \quad \forall x \in \boldsymbol{x}. \tag{9.10}$$

Der Leser sollte verstanden haben, dass man mit (9.10) für beliebiges N stets eine garantierte Einschließung der coth-Funktionswerte erhält. Aber natürlich muss der Polynomgrad N noch geeignet gewählt werden, denn bei zu großem N erhält man unnötig große Laufzeiten und bei zu kleinem Polynomgrad wird die durch `stagprec` verlangte Genauigkeit nicht erreicht.

In $A_1 = [\texttt{MinReal}, 2^{-67})$, mit $\texttt{Inf(dx)} \in A_1$, ist die Berechnung eines geeigneten Polynomgrades N besonders einfach, denn der Funktionswert $\coth(\texttt{Inf(dx)})$ wird nach (9.7) mit $\texttt{ex} := \texttt{expo(Inf(dx))}$ gut approximiert durch $(\texttt{Inf(dx)})^{-1} \approx 2^{-\texttt{ex}}$. Bei einer gewünschten Genauigkeit von $53 \cdot \texttt{stagprec}$ binären Ziffern, sollte daher der durch $b_n(x)$ gegebene absolute Fehler nicht größer sein als $2^{-\texttt{ex}-53\cdot\texttt{stagprec}}$, wobei $b_n(x)$ an der Stelle $x = \texttt{Sup(dx)}$ auszuwerten ist. Bezeichnen wir dann mit $\texttt{expo}(b_n)$ den Exponenten von $b_n(x)$ zur Basis 2, so muss in einer Schleife $n = 1, 2, \ldots$ solange erhöht werden, bis für $n = n_0$ gilt:

$$\texttt{expo}(b_{n_0}) < \texttt{ex} - 53 \cdot \texttt{stagprec} =: \texttt{m} \tag{9.11}$$

Das Maximum des absoluten Fehlers ist dann gegeben durch $\alpha = b_{n_0}(\texttt{Sup(dx)})$, und $N := n_0 - 1$ ist der gesuchte Polynomgrad. Testrechnungen zeigen, dass man bei nicht zu breiten Argumentintervallen $\boldsymbol{x}$, mit $\texttt{Inf}(\boldsymbol{x}) \in A_1$ und $\texttt{Sup}(\boldsymbol{x}) < 4 \cdot 10^{-19}$ bei $\texttt{stagprec} \leq 19 =: \texttt{stagmax}$ die Abbruchbedingung (9.11) bis zum maximalen Wert $N = 7$ stets erfüllen kann. Bei zu großem $\texttt{Sup}(\boldsymbol{x})$, d.h. $N > 7$, wird die obige Schleife abgebrochen und $\coth(\boldsymbol{x})$ wird in nur einfacher Genauigkei t eingeschlossen. Aus Laufzeitgründen wird die `do-while`-Schleife in einfacher Genauigkeit ausgeführt; beachten Sie jedoch, dass wir mit `r` vom Typ *interval* eine garantierte Oberschranke des Faktors x^{2n-1} aus (9.9) erhalten und damit nach der Schleife mit Hilfe der Anweisung `r2 = r*abs(r2)/3` entsprechend (9.8) eine Einschließung des absoluten Fehlers berechnen können. In der folgenden `for`-Schleife wird dann nach dem Hornerschema mit `y` eine Einschließung des Polynoms $P_N(u)$ aus (9.7) berechnet, und die letzte Anweisung `y = y +`

[2]Im Quelltext ab Seite 377 wird α realisiert durch `Sup(r2)`

`interval(-Sup(r2),Sup(r2))` berücksichtigt noch den Approximationsfehler. $N = 6$ ist der maximale Polynomgrad.

9.3.3 Fehlerabschätzung in A_4

Im Falle $\texttt{Inf}(\boldsymbol{x}) \geq 353$ wird $\coth(\boldsymbol{x})$ durch `y` eingeschlossen, mit $\texttt{Sup}(\texttt{y}) = 1 + 2^{-1017}$ und $\texttt{Inf}(\texttt{y}) = 1$. Da die Funktion $\coth(x)$ für positive x monoton fällt, muss daher nach (9.6) nur gezeigt werden:

$$1 + \frac{2}{e^{2\cdot 353} - 1} \leq 1 + 2^{-1017} \quad \Longleftrightarrow \quad \frac{1}{e^{706} - 1} \leq 2^{-1018},$$

wobei die letzte Ungleichung z.B. mit dem Algebra-System *Maple* leicht bestätigt werden kann.

Hinweise zum Algorithmus:

1. Da in den Teilbereichen A_2, A_3 die coth-Funktion mit identischen Funktionstermen intervallmäßig ausgewertet wird, sind die jeweils berechneten Intervalle `y` nach [2, Seite 31] garantierte Einschließungen der Wertebereiche $\coth(\boldsymbol{x})$, so dass eine Fehlerabschätzung nicht notwendig ist.

2. Die Auswertung von $1+2/(e^{2x}-1)$ in A_3 erfordert nur kurze Laufzeiten, da die Exponentialfunktion nur einmal zu berechnen ist. Bei zu großen Argumenten liefert e^{2x} einen Overflow, so dass $\texttt{Inf}(\boldsymbol{x}) \leq 535$ verlangt wird. Bei zu kleinen Argumenten tritt bei der Auswertung des Nenners $(e^{2x} - 1)$ eine zu starke Auslöschung auf, die durch die Forderung $\texttt{Inf}(\texttt{dx}) \geq 2$ vermieden wird.

3. Die Auswertung des Quotienten $\cosh(\boldsymbol{x})/\sinh(\boldsymbol{x})$ im Bereich A_2 verhindert die oben beschriebene Auslöschung. Allerdings erfordern die beiden hyperbolischen Funktionen eine deutlich größere Laufzeit. Wählt man z.B. $\boldsymbol{x} = [2^{-1020}, 2^{-1020}]$, so liefert der obige Quotient auch mit $\texttt{stagprec} = 19$ eine Einschließung von $\coth(\boldsymbol{x})$ in nur einfacher Genauigkeit, d.h. mit nur 16 korrekten Dezimalziffern, da das Intervall $\sinh(\boldsymbol{x})$ selbst kein Punktintervall mehr sein kann. Wegen der Beziehung $\sinh(\boldsymbol{x}) \approx \boldsymbol{x} = [2^{-1020}, 2^{-1020}]$ kann daher die Einschließung von $\sinh(\boldsymbol{x})$ nur noch mit einer Genauigkeit von $1074 - 1020 = 54$ binären Ziffern erfolgen, und auch der Quotient $\cosh(\boldsymbol{x})/\sinh(\boldsymbol{x})$ lässt sich dann natürlich mit keiner größeren Genauigkeit einschließen. Aus diesem Grund kommt die Identität $\coth(x) \equiv \cosh(x)/\sinh(x)$ im Bereich A_2 nur für $\texttt{Inf}(\texttt{dx}) \geq 2^{-67}$ zur Anwendung.

4. Beachten Sie, dass in A_1 der relative Durchmesser $d > 0$ eines Intervalls $\boldsymbol{x}$ umso größer wird, je kleiner $\texttt{Inf}(\boldsymbol{x})$ gewählt wird. So ist der relative Durchmesser von $\boldsymbol{x} = [\texttt{MinReal}, \texttt{MinReal} + \texttt{minreal}]$ gegeben durch $d = \texttt{minreal}/\texttt{MinReal} =$

2^{-52}, so dass $\coth(\boldsymbol{x})$ mit einer Genauigkeit von höchstens 52 binären Stellen eingeschlossen werden kann. Vergleichen Sie dazu bitte auch die numerischen Ergebnisse im folgenden Unterabschnitt.

9.3.4 Numerische Ergebnisse

Zu einem vorgegebenen Maschinenintervall `x` vom Typ *l_interval* wird durch die Anweisung `y = coth(x)` ein Intervall `y`, ebenfalls vom Typ *l_interval*, berechnet, das den Wertebereich $W := \{y \,|\, y = \coth(t),\ t \in \mathtt{x}\}$ garantiert einschließt, d.h. $W \subseteq \mathtt{y}$. Mit $1 \leq \mathtt{stagprec} \leq 19$ wird die Präzision der Staggered Correction Arithmetik vorgegeben, wobei etwa $16 \cdot \mathtt{stagprec}$ dezimale Ziffern zur Verfügung stehen.

Mit $\mathtt{x} = [\mathtt{MinReal}, \mathtt{MinReal}+\mathtt{minreal}] = [2^{-1022}, 2^{-1022}+2^{-1074}]$ und $\mathtt{stagprec} = 2$ erhält man für W die Einschließung:

```
y = [4.49423283715578827244303082593270E+0307,
     4.49423283715579176516357251169238E+0307]
```

Mit $\mathtt{x} = [2^{-1022}, 2^{-1022}]$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [4.49423283715578976932326297697256183404494244735576643183575202894
       33168951375240783177119330601884005280028469967848339414697442203
       60415562321185765986853109444197335621637131907555490031152352986
       32707380212514422095376705856157203684782776352068092908376276711
       46574559986811484619929076208839082406056034E+0307,
     4.49423283715578976932326297697256183404494244735576643183575202894
       33168951375240783177119330601884005280028469967848339414697442203
       60415562321185765986853109444197335621637131907555490031152352986
       32707380212514422095376705856157203684782776352068092908376276711
       46574559986811484619929076208839082406056035E+0307]
```

Mit $\mathtt{x} = [2^{-68}, 2^{-68}]$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [2.95147905179352825856000000000000000000011293772630057337854244300
       00090618928273518793777565585261920734110221475276052214024369927
       43346593204320639044964461169105768764715669116089041461959884431
       08042745535018667557951659180923452944465783788918816201451917602
       15376424124817857859048270648862301747247756E+0020,
     2.95147905179352825856000000000000000000011293772630057337854244300
       00090618928273518793777565585261920734110221475276052214024369927
       43346593204320639044964461169105768764715669116089041461959884431
       08042745535018667557951659180923452944465783788918816201451917602
       15376424124817857859048270648862301747247757E+0020]
```

Mit $\mathtt{x} = [2^{-66}, 2^{-66}]$ und $\mathtt{stagprec} = 10$ erhält man für W die Einschließung:

```
y = [7.37869762948382064640000000000000000000045175090520229351416977200
       03624757130940699892641974567629269830541744176673416976503886407
       64222427131541016420536176755 6E+0019,
     7.37869762948382064640000000000000000000045175090520229351416977200
       03624757130940699892641974567629269830541744176673416976503886407
       64222427131541016420536176755 7E+0019]
```

Mit $\mathtt{x} = [1.75, 1.75]$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [1.06227531851559957239027872176814563905036583341092566420115342510
       63992205212286191431134193631642728710038329597407475974197462958
       38945152415031446750329593933799795152628418036911957932110933443
       94147584718706116529133464437733016938689736119912301290870003924
       85191965488067550622256328876097074086916859E+0000,
     1.06227531851559957239027872176814563905036583341092566420115342510
       63992205212286191431134193631642728710038329597407475974197462958
       38945152415031446750329593933799795152628418036911957932110933443
       94147584718706116529133464437733016938689736119912301290870003924
       85191965488067550622256328876097074086916861E+0000]
```

Mit $[2.1, 2.1] \subset \mathtt{x} = \mathtt{l\_interval}(21)/10$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [1.03044773499000761867433049008535777195684143426744778480029036685
       48867777134767308562865250144820203727067139923575933454868849239
       81959873105405360199424978882877650497437518709130018056152628009
       34173035827407074439513238504972745132715864655800267963278442911
       87973020979982940861663765903635143125902169E+0000,
       1.030447734990007618674330490085357771956841434267447784800290366
       85488677771347673085628652501448202037270671399235759334548688492
       39819598731054053601994249788828776504974375187091300180561526280
       09341730358274070744395132385049727451327158646558002679632784429
       1187973020979982940861663765903635143125902170E+0000]
```

Mit $\mathtt{x} = [352, 352]$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [1.0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000E+0000,
     1.0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000000000000000000000000000
       0000000000000000000000000000000000000000001E+0000]
```

Mit $\mathtt{x} = [-20000, -354]$ und $\mathtt{stagprec} = 10$ erhält man für W die Einschließung:

```
y = [-1.0000000000000000000000000000000000000000000000000000000000000000
        0000000000000000000000000000000000000000000000000000000000000000
        0000000000000000000000000000001E+0000,
     -1.0000000000000000000000000000000000000000000000000000000000000000
        0000000000000000000000000000000000000000000000000000000000000000
        0000000000000000000000000000000E+0000]
```

9.3.5 Quelltext

```
// Programm coth.cpp zur Auswertung der Funktion
// coth(x) für Intervallargumente vom Typ l_interval

#include <imath.hpp>
#include <l_imath.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

// Fields for coth-function
int  ex_coth[8] = { -1,-5,-8,-12,-15,-18,-22,-25 };
int  c_cothN[8] = { 1,-1,2,-1,2,-1382,4,-3617 };
real c_cothD[8] = { 1,15,315,1575,31185,212837625,  // all components
                    6081075,54273594375.0 };        // exactly stored!

l_interval Coth(const l_interval & x) throw()
// Coth(x); Blomquist 17.04.04;
{
    l_interval t,s,c,y;
    interval dx = _interval(x),
             einfachgenau,r,r2;
    int stagsave = stagprec,ex,m,N,k,
        stagmax = 19;
    bool neg;
    einfachgenau = coth(dx); // produces all necessary error messages!

    if (stagprec == 1) y = coth(dx);
    else
    {
        if (stagprec>stagmax) stagprec = stagmax;
        neg = Sup(dx)<0.0;
        t = x;
        if (neg) { t = -t; dx = -dx; } // Inf(t),Inf(dx) > 0;
        ex = expo(Inf(dx));
        if (ex<-66) { // Laurent series if Inf(dx)<2^(-67)
            m = -ex - 53*stagprec;
            r = interval(Sup(dx)); r2 = r*r;
            N = 0;
```

```
            do {
                N++;
                if (N>7) { y = einfachgenau; goto Fertig; }
                r = r*r2;
                k = expo(Sup(r)) + ex_coth[N];
            } while (k>m);
            r2 = interval(c_cothN[N])/c_cothD[N];
            r2 = r*abs(r2)/3;  // r2: inclusion of the absolute error
            N--; // Polynomial degree
            y = l_interval(c_cothN[N])/c_cothD[N];
            s = t*t;
            for (k=N-1; k>=0; k--)
                y = y*s + l_interval(c_cothN[k])/c_cothD[k];
            y = (y*t)/3 + 1/t;
            y = y + interval(-Sup(r2),Sup(r2)); // with approx. error
        } else if (ex < 2) {
            if (stagprec<stagmax) stagprec++; // stagprec <= 19
            c = cosh(t);
            s = sinh(t);
            y = c/s;
        } else if (Inf(dx)<353.0) {
            if (stagprec<stagmax) stagprec++; // stagprec <= 19
            times2pown(t,1);
            y = 1.0 + 2.0/(exp(t)-1.0);
        } else { // Inf(dx) >= 353.0
            y = l_interval(1.0) + comp(0.5,-1016);
            SetInf(y,1.0);
        }
        if (_interval(1.0) <= y)  SetInf(y,1.0);
        if (neg) y = -y;
    Fertig:     stagprec = stagsave; // restore old stagprec value
        y = adjust(y);  // adjust precision of y to old stagprec value
        y = y & einfachgenau;
    }
    return y;
} // coth

int main()
{
```

```
    l_interval x,y;
    real r1,r2;
    stagprec = 2;  // stagmax = 19

    while (1) {
        cout << "real r1 = ?" << endl;   cin >> r1;
        cout << "real r2 = ?" << endl;   cin >> r2;
        x = l_interval(r1,r2);  // x = l_interval(21)/10;
        y = Coth(x);
        cout << SetDotPrecision(16*stagprec,16*stagprec)
             << Scientific << "Coth(x) = " << y << endl << endl;
    };
}
```

Hinweise:

1. Zur Vermeidung von Namenskonflikten beim Übersetzen des obigen **C-XSC** Programms `coth.cpp` wurde die hyperbolische Cotangens-Funktion mit `Coth` bezeichnet. Die gleiche Funktion wird im **C-XSC** System in `l_imath.hpp` unter dem Namen `coth` deklariert.

2. Der dezimale Eingabewert z.B. für `r1` wird zur nächsten binären Rasterzahl in der durch `stagprec` vorgegebenen Präzision gerundet und anschließend in der Variablen `r1` gespeichert. Gibt man daher für `r1,r2` die gleichen Dezimalwerte ein, so erhält man mit der Anweisung `x = l_interval(r1,r2)` für `x` stets binäre Punktintervalle. Beachten Sie jedoch, dass wir mit der Eingabe von 2.1 für `r1` und `r2` durch `x = l_interval(r1,r2)` zwar ein Punktintervall erhalten, das jedoch die Zahl 2.1 nicht enthält, da 2.1 nicht durch eine endliche Binärzahl dargestellt werden kann. Es gilt daher: `r1` $=$ `r2` $\neq 2.1 \notin$ `x`. Erst die Anweisung
   `x = l_interval(21)/10` liefert ein Intervall `x`, das 2.1 mit der gewünschten Präzision tatsächlich und sogar optimal einschließt und daher auch nicht punktförmig sein kann.

3. Wird bei einem zu breiten Argumentintervall $\boldsymbol{x}$ der Intervalldurchmesser von `y` wegen der unvermeidbaren Intervallüberschätzungen größer als der Durchmesser von `einfachgenau`, so liefert der Durchschnitt `y = y & einfachgenau` mit `y` eine optimale Einschließung von $\coth(\boldsymbol{x})$.

4. In den Teilbereichen A_2, A_3 werden die Funktionsterme $\cosh(\boldsymbol{x})/\sinh(\boldsymbol{x})$ bzw. $1 + 2/(e^{2\boldsymbol{x}} - 1)$ wegen `stagprec++` zunächst in erhöhter Präzision berechnet, um damit die Wertebereiche in der ursprünglichen Präzision möglichst genau einschließen zu können.

9.4 $\sqrt{1-x^2}$

Zu einem vorgegebenen und nicht zu breiten Argumentintervall $\boldsymbol{x}$ vom Typ *l_interval* ist gesucht eine garantierte und möglichst enge Einschließung der Funktionswerte $f(x) = \sqrt{1-x^2}$ für alle reellen $x \in \boldsymbol{x}$, mit $\boldsymbol{x} \subseteq [-1, +1]$.

9.4.1 Der Algorithmus

Wegen $f(-x) \equiv f(x)$ sind die Wertebereiche für die beiden Argumentintervalle $\boldsymbol{x}$ und $\boldsymbol{z} = \texttt{abs}(\boldsymbol{x})$ identisch, so dass man sich wie auf Seite 323 auf das Argumentintervall $\boldsymbol{z}$ beschränken kann. Auf den ersten Blick scheint eine Einschließung von

$$W_z := \left\{ y \,|\, y = \sqrt{1-x^2}, x \in \boldsymbol{z} \right\}$$

trivial zu sein, denn bei der intervallmäßigen Auswertung von $1 - \boldsymbol{z} \cdot \boldsymbol{z}$ erhält man mit $\boldsymbol{w} := 1 \ominus \boldsymbol{z} \odot \boldsymbol{z}$ automatisch eine garantierte Einschließung von $1 - \boldsymbol{z} \cdot \boldsymbol{z}$ und mit $\boldsymbol{y} := \texttt{sqrt}(\boldsymbol{w})$ die gesuchte, garantierte Einschließung von $W_z \subseteq \boldsymbol{y}$. Allerdings werden diese Einschließungen selbst für Punktintervalle $\boldsymbol{z}$ mit $\texttt{Sup}(\boldsymbol{z}) \to +1$ immer gröber, und mit $\boldsymbol{z} = [1 - \texttt{MinReal}, 1 - \texttt{MinReal}]$ wird die Einschließung $\boldsymbol{y}$ auch bei maximaler Präzision, d.h. $\texttt{stagprec} = 19$, nur noch mit einer Genauigkeit von etwa 16 Dezimalstellen berechnet. Der Grund für die Intervallaufblähungen von $\boldsymbol{y}$ sind die bei der Berechnung von $\boldsymbol{w}$ auftretenden Auslöschungseffekte und die schon erwähnte Tatsache, dass $\texttt{minreal} = 2^{-1074}$ auch bei der Staggered Correction Arithmetik die kleinste zur Verfügung stehende positive Zahl ist. Auch die Zerlegungen $1 - \boldsymbol{z}^2 = (1 - \boldsymbol{z}) \cdot (1 + \boldsymbol{z})$ oder $1 - \boldsymbol{z}^2 = 2\boldsymbol{\delta} - \boldsymbol{\delta}^2$, mit $\boldsymbol{\delta} := 1 - \boldsymbol{z}$ liefern keine Verbesserung der Genauigkeit.

Die beschriebenen Auslöschungseffekte und die dadurch bedingten Ungenauigkeiten lassen sich jedoch durch die folgende Skalierung völlig vermeiden:

$$\sqrt{1-\boldsymbol{z}^2} \equiv 2^{-m} \cdot \sqrt{2^{2m} - (2^m \cdot \boldsymbol{z}) \cdot (2^m \cdot \boldsymbol{z})}, \quad m = +511$$

Die Multiplikation von $\boldsymbol{z}$ mit 2^m erfolgt mit Hilfe der Funktion `times2pown(z,511)`, und der Radikand $2^{2m} - (2^m \cdot \boldsymbol{z}) \cdot (2^m \cdot \boldsymbol{z})$ hat mit $\boldsymbol{z} = [1 - \texttt{MinReal}, 1 - \texttt{MinReal}]$ die Größenordnung 2, so dass zu seiner Berechnung immer noch etwa 320 Dezimalstellen zur Verfügung stehen. Die Quadratwurzel kann damit hinreichend genau berechnet werden, und die Rückskalierung mit 2^{-m} liefert die bestmögliche Einschließung $\boldsymbol{y}$ des Wertebereichs $W_z \subseteq \boldsymbol{y}$. Beachten Sie bitte, dass die Skalierungsoperationen mit `times2pown(...)` im Vergleich zu normalen Multiplikationen laufzeitmäßig praktisch keine Rolle spielen und dass bei diesen Operationen im denormalisierten Bereich die Rundungen wie bei Intervallmultiplikationen durchgeführt werden, d.h. auch bei der Rückskalierung werden garantierte Einschließungen berechnet.

Da der beschriebene Algorithmus nur Intervalloperationen enthält und damit automatisch Einschließungen erzeugt, ist eine Fehlerabschätzung nicht erforderlich.

9.4.2 Numerische Ergebnisse

Zu einem vorgegebenen Intervall `x` vom Typ *l_interval* wird durch die Anweisung `y = sqrt1mx2(x)` ein Intervall `y`, ebenfalls vom Typ *l_interval*, berechnet, das den Wertebereich $W := \{y \,|\, y = \sqrt{1-t^2},\ t \in \mathtt{x}\}$ garantiert einschließt, d.h. $W \subseteq \mathtt{y}$. Mit $1 \leq \mathtt{stagprec} \leq 19$ wird die Präzision der Staggered Correction Arithmetik vorgegeben, wobei etwa $16 \cdot \mathtt{stagprec}$ dezimale Ziffern zur Verfügung stehen.

Mit $\mathtt{x} = [1 - \mathtt{MinReal}, 1 - \mathtt{MinReal}] = [1 - 2^{-1022}, 1 - 2^{-1022}]$ und $\mathtt{stagprec} = 19$ erhält man für W die optimale Einschließung:

```
y = [2.109537322972599782530539911842895235336244293902247636845747232692
       756255030155342372892684752321922282711136620473376634838896521 6
       458672546864741758162867321681010973998431391316556670986324486 50
       329896264659248533381803772251333010872912378685693726166857589 51
       119049088353071428518280782761370124419683 44E-0154,
     2.109537322972599782530539911842895235336244293902247636845747232692
       756255030155342372892684752321922282711136620473376634838896521 6
       458672546864741758162867321681010974002877982129127889883913605 63
       911295499787802355734696060234360125123520335228881602845627442 23
       666288402534762251108395378544769228351718 03E-0154]
```

Mit $\mathtt{x} = [1 - \mathtt{MinReal}, 1 - \mathtt{MinReal}] = [1 - 2^{-1022}, 1 - 2^{-1022}]$ und $\mathtt{stagprec} = 19$ erhält man ohne Skalierung für W die Einschließung:

```
y = [2.109537322972599665427694548328108335535766791161320983246813445 64
       917304069156992771789253654031816881166466294434397226457714443 33
       467656895670911096920506443399093479235361543048202171941892532 17
       496262491416284447208736580315100945834722392284147126927002792 39
       874758756431403743932455009812614649362992 14E-0154,
     2.109537322972599782530539911842925834264997810180683855498729981 70
       234418321340964881711631484810044747185361344781723166929985335 90
       907526237670314422599181862380937312959179707795307535255464448 84
       933565487115785119496975452575316186286912357772969873742937070 31
       486943466421188908728001060626900703064043 29E-0154]
```

Mit $\mathtt{x} = [0.5, 0.5]$ und $\mathtt{stagprec} = 19$ erhält man für W die Einschließung:

```
y = [8.660254037844386467637231707529361834714026269051903140279034897 25
       966508454400018540573093378624287837813070707703351514984972547 49
       947623940582775604718682426404661595115279103398741005054233746 16
       325076561716334516614433253361273344609189856135235658301839307 94
       009524993268689929694733825173753288025378 30E-0001,
     8.660254037844386467637231707529361834714026269051903140279034897 25
       966508454400018540573093378624287837813070707703351514984972547 49
       947623940582775604718682426404661595115279103398741005054233746 16
       325076561716334516614433253361273344609189856135235658301839307 94
       009524993268689929694733825173753288025378 31E-0001]
```

9.4.3 Quelltext

```
// Programm sqrt1mx2_l.cpp zur Auswertung der Funktion
// sqrt(1-x^2) für x vom Typ l_interval

#include <imath.hpp>
#include <l_imath.hpp>
#include <iostream>

using namespace std;
using namespace cxsc;

l_interval Sqrt1mx2(const l_interval& x)
// sqrt(1-x^2);  Blomquist, 13.04.04;
{
    int stagsave = stagprec,
        stagmax = 19;
    l_interval y,z = abs(x);
    l_real r1,r2;
    interval dx = interval(z),
             einfachgenau;
    einfachgenau = sqrt1mx2(dx);
    if (stagprec>stagmax) stagprec = stagmax;

    if (stagprec == 1) y = sqrt1mx2(dx); // simple interval
    else {
        y = comp(0.5,1023); // y = 2^(+1022)
        times2pown(z,511);
        y = sqrt(y-z*z);
        times2pown(y,-511);
    }
    stagprec = stagsave;
    y = adjust(y);
    y = y & einfachgenau;
    return y;
} // Sqrt1mx2

int main()
{
    stagprec = 19;
    l_interval x,y;
```

```
    real r;

    while (1) {
        cout << "real r = ?" << endl;   cin >> r;
        x = l_interval(r);  // x = x - MinReal;
        y = Sqrt1mx2(x);
        cout << SetDotPrecision(16*stagprec,16*stagprec)
             << Scientific << "sqrt1mx2(x) = " << y << endl << endl;
    };
}
```

Hinweise:

1. Zur Eingabe des Punktintervalls $[1 - \texttt{MinReal}, 1 - \texttt{MinReal}]$ ist obige `while`-Schleife wie folgt abzuändern:

   ```
   while (1) {
       cout << "real r = ?" << endl;   cin >> r;
       x = l_interval(1.0) - MinReal;
       y = Sqrt1mx2(x);
       cout << SetDotPrecision(16*stagprec,16*stagprec)
            << Scientific << "sqrt1mx2(x) = " << y << endl << endl;
   };
   ```

   Die Zeile `cout << "real r = ?" << endl;  cin >> r;` verhindert jetzt nur das Durchlaufen der Schleife und hat sonst keine Bedeutung!

2. Zur Vermeidung von Namenskonflikten beim Übersetzen des obigen **C-XSC** Programms `sqrt1mx2_l.cpp` wurde der Funktionsname `Sqrt1mx2` gewählt. Die gleiche Funktion $\sqrt{1-x^2}$ wird im **C-XSC** System in `l_imath.hpp` unter dem Namen `sqrt1mx2` deklariert.

3. Vergleichen Sie bitte auch den Hinweis 3 auf Seite 379.

Kapitel 10

Minimumeinschließung konvexer Funktionen

10.1 Problemstellung

Die reelle Funktion

$$f\ :\ \mathbb{D}_f = [x] := [x_1, x_2]\ \longrightarrow\ \mathbb{R}, \qquad x_1, x_2 \in \mathbb{R}$$

sei in $\mathbb{D}_f$ streng konvex [17] und besitze im Innenpunkt $x_0 \in]x_1, x_2[$ ein lokales Minimum. Wertet man die reelle Funktion f für eine Maschinenzahl $x \in S(2,53)$ auf einem Rechner aus, so erhält man den i.a. fehlerbehafteten Maschinenwert

$$\widetilde{f}(x) = f(x) \cdot (1 + \varepsilon_f),$$

und für den relativen Fehler ε_f gilt: $\quad |\varepsilon_f| \leq \varepsilon(f), \quad x \in \mathbb{D}_f \cap S(2,53)$.
In der numerischen Praxis sind i.a. weder die Minimumstelle x_0 noch $f(x_0)$ bekannt; außerdem werden x_0 und $f(x_0)$ nur in seltenen Ausnahmefällen Maschinenzahlen sein; aber selbst dann kann man wegen unvermeidbarer Rundungsfehler nicht erwarten, dass die Maschinenergebnisse einer entsprechenden Rechnung mit x_0 bzw. $f(x_0)$ übereinstimmen werden. Gesucht ist damit ein Algorithmus, der garantierte und möglichst enge Einschließungen von x_0 und $f(x_0)$ liefert:

$$x_0 \in \mathtt{xe} = [x_1, x_2] = ? \qquad f(x_0) \in f(\mathtt{xe}) \subset \mathtt{ye} = [y_1, y_2] = ?$$

Genau diese Einschließungen `xe,ye` werden benötigt, wenn zu einer gegebenen Funktion f mit relativen Extrema und bekannter Fehlerschranke $\varepsilon(f)$ eine entsprechende Intervall-Version zu implementieren ist.

Beachten Sie bitte, dass die Intervalle `xe,ye` mit den Werkzeugen der globalen Optimierung nicht berechnet werden können, da f dann auf die in der Differentiationsarithmetik bereitgestellten Funktionen beschränkt wäre und zusätzlich zweimal

differenzierbar sein müsste. Wir setzen jedoch lediglich voraus, dass die Funktion f in ihrem Definitionsbereich $\mathbb{D}_f$ streng konvex ist und in $x_0 \in]\,x_1, x_2\,[$ ein lokales Minimum besitzt. Sowohl für x_0 als auch für das Minimum $f(x_0)$ kann man unter diesen Voraussetzungen garantierte Einschließungen berechnen, wenn man f nur für endliche viele **Punkt**–Argumente auswertet. In der folgenden Abbildung bedeuten z.B. $\underline{f(a)}$ und $\overline{f(a)}$ Unter– bzw. Oberschranken für den exakten Funktionswert $f(a)$, wobei die genannten Maschinenschranken bei bekannter Fehlerschranke $\varepsilon(f)$ nach (3.19) und (3.20) von Seite 78 berechnet werden können, wenn man dort die Argumentintervalle $[\widetilde{x}_1, \widetilde{x}_2]$ als Punktintervalle auffasst.

10.2 Einschließung der Minimumstelle x_0

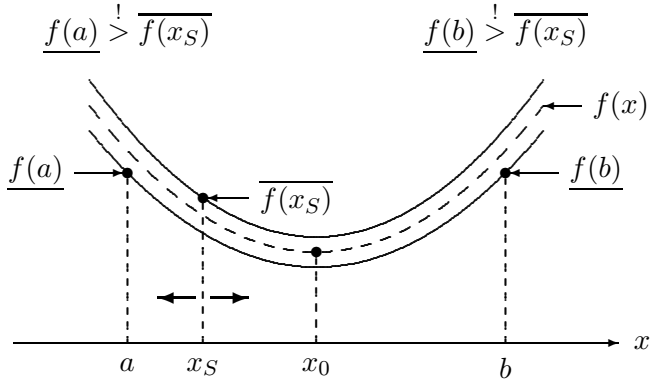


Abbildung 10.1: Einschließung von x_0

Nach Abb. 10.1 beginnt man mit dem Startwert x_S und läuft zunächst nach links, bis im Punkt $a < x_S$ die Bedingung $\overline{f(x_S)} < \underline{f(a)}$ erfüllt ist. Anschließend geht man von x_S aus schrittweise nach rechts, bis der Fall $\overline{f(x_S)} < \underline{f(b)}$ eingetreten ist. Elementare Rechnungen ergeben dann:

$$f(x_S) < f(a) \qquad \text{und} \qquad f(x_S) < f(b),$$

und wegen $a < x_S < b$ gilt dann bei einer konvexen Funktion mit **nur einer** lokalen Minimumstelle x_0:

$$a < x_0 < b \tag{10.1}$$

Mit dem neuen, verbesserten Startwert $x_S := 0.5 \cdot (a + b)$ wiederholt man die Rechnungen solange, bis sich der Intervalldurchmesser ($b-> a$) nicht weiter verkleinert. Damit erhält man für $x_0 \in [a, b]$ eine garantierte Einschließung, die um so enger

ist, je besser die exakten Funktionswerte durch die jeweiligen Maschinenschranken eingeschlossen werden können.

10.3 Einschließung des Minimums $f(x_0)$

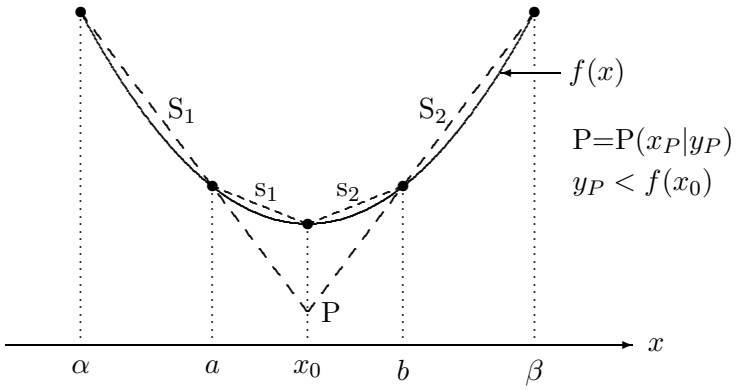


Abbildung 10.2: Einschließung des Minimums f(x_0)

In Abb. 10.2 schneiden sich die Sekanten s_1, s_2 nach Voraussetzung im Minimumpunkt $(x_0|f(x_0))$. Wegen $x_0 < b < \beta$ und weil $f(x)$ nach Voraussetzung konvex ist, gilt:

$$\text{Steigung}(S_2) > \text{Steigung}(s_2); \quad \text{analog folgt:} \quad \text{Steigung}(S_1) < \text{Steigung}(s_1).$$

Sind dann x_P, y_P die Koordinaten des Sekantenschnittpunkts $P(x_P|y_P)$, so zeigen einfache Rechnungen: $y_P < f(x_0)$. Damit gilt für den lokalen Minimumwert f(x_0) die Einschließung:

$$y_P < f(x_0) < \text{Max}\{f(a), f(b)\} \tag{10.2}$$

Da im Rechner nicht $f(x)$ sondern nur die Schranken $\underline{f(x)}$ und $\overline{f(x)}$ zur Verfügung stehen, kann nicht S_2 selbst, sondern nur die Sekante $\widetilde{S}_2$ durch die Punkte $(b|\underline{f(b)})$ und $(\beta|\overline{f(\beta)})$ durch gerichtetes Runden berechnet werden, vergl. Abb. 10.3.
Es gilt dann:

$$\text{Steigung}(\widetilde{S}_2) > \text{Seigung}(S_2),$$

und weil die Sekante $\widetilde{S}_2$ im Vergleich zu S_2 durch den tiefer gelegenen Punkt $(b|\underline{f(b)})$ läuft, erhält man für die $\widetilde{y}_P$–Koordinate des entsprechend tiefer liegenden Schnittpunktes $\widetilde{P}(\widetilde{x}_P|\widetilde{y}_P)$ der beiden Sekanten $\widetilde{S}_1, \widetilde{S}_2$ die Ungleichung: $\widetilde{y}_P < y_P$. Um eine garantierte Einschließung von $f(x_0)$ zu erhalten, muss $\widetilde{y}_P$ im Programm durch

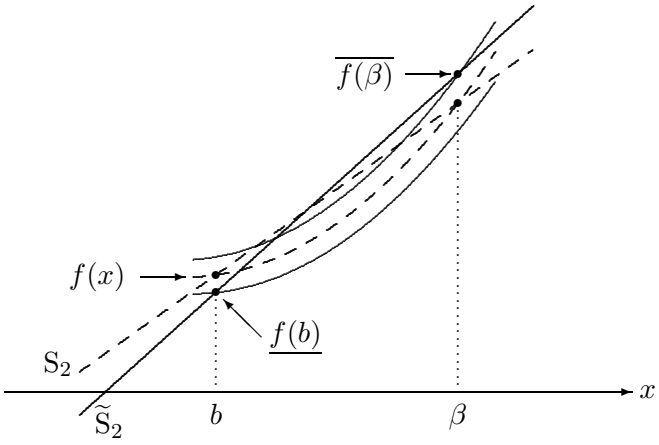


Abbildung 10.3: $\widetilde{S}_2$ liegt tiefer und ist steiler als S_2

gerichtetes Runden berechnet werden, und in (10.2) sind $f(a)$, $f(b)$ durch ihre Oberschranken zu ersetzen; man erhält damit die Einschließung:

$$\widetilde{y}_P < f(x_0) < \mathrm{Max}\{\overline{f(a)}, \overline{f(b)}\}. \tag{10.3}$$

Abschließend sei noch bemerkt, dass zur Berechnung der beschriebenen Einschließungen von x_0 und $f(x_0)$ keine Ableitungen der Funktion f benötigt werden, wodurch der Rechenaufwand begrenzt bleibt! Beachten Sie bitte, dass man wegen (10.3) nicht nur eine Einschließung von $f(x_0)$ sondern auch von $f([a,b])$ erhält, wenn $[a,b]$ eine Einschließung der Minimumstelle x_0 ist.

10.4 Anwendungen

Mit dem C-XSC Programm `Min-Incl` kann man garantierte Einschließungen der Minimumstelle x_0 und des Minimumwertes $f(x_0)$ einer streng konvexen Funktion berechnen. Die Funktion $f(x)$ ist dabei am Anfang des Quelltextes zu definieren. Ein Beispiel zur Implementierung findet man zusammen mit dem kompletten Quellcode im Anhang B ab Seite 407. Für die beiden reellen Funktionen

$$f(x) = \begin{cases} x^2 & x \le 0 \\ x^2 \cdot \ln(x) & x > 0 \end{cases} \qquad \text{und} \qquad f(x) = e^{-x} \cdot \ln(1+x),\ x > -1$$

wird in den beiden folgenden Unterabschnitten die Einschließung von x_0 und $f(x_0)$ berechnet, dabei ist der relative Fehler $\varepsilon(f)$ von $f(x)$ jeweils vorher abzuschätzen.

10.4.1 $f(x) = x^2 \cdot \ln(x),\ x > 0$

Die zu untersuchende Funktion $f(x)$ wird für alle $x \in \mathbb{R}$ definiert durch:

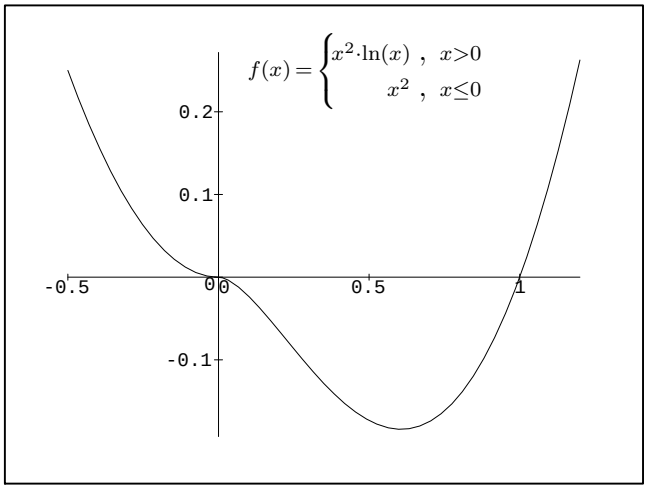
$$f(x) = \begin{cases} x^2 & : \quad x \leq 0 \\ x^2 \cdot \ln(x) & : \quad x > 0 \end{cases}$$

und elementare Rechnungen liefern:

$$\text{Minimumstelle: } x_0 = \frac{1}{\sqrt{e}}; \qquad \text{Minimum: } f(x_0) = \frac{-1}{2e} \leq f(x)\,;$$

$$(x < 0) \quad \text{oder} \quad \left(x > \frac{1}{e\sqrt{e}}\right) \quad \rightsquigarrow \quad f \text{ ist streng konvex;}$$

Aus dem nachfolgenden Graph der Funktion erkennt man weiter, dass f nicht im ganzen Definitionsbereich konvex ist:



Berechnung der Fehlerschranke $\varepsilon(f)$:

Wegen $x_0 > 0$ muss der relative Fehler nur für $x > 0$ abgeschätzt zu werden. Dabei ist zu beachten, dass für $x \to 0$ die Maschinenwerte $x \odot x$ in den denormalisierten Bereich fallen und damit die relativen Fehler viel zu groß werden. Wir beschränken die Fehlerabschätzung daher auf den Bereich $[0.25, \texttt{pred}(1.0)]$, da für $x \leq 1/(e\sqrt{e}) = 0.2231\ldots$ die Funktion $f(x)$ nicht mehr konvex ist und damit das

Programm `Min-Incl` zumindest theoretisch nicht mehr funktionieren muss. Zusätzlich verlangen wir $x \leq \texttt{pred}(1.0)$, da $f(x)$ bei $x = 1$ eine Nullstelle besitzt und die automatische Abschätzung des relativen Fehlers an dieser Stelle $x = 1$ nicht möglich ist. Wegen $\widetilde{\ln}(1.0) = 0$ kann der relative Fehler für $x = 1$ auf Null gesetzt werden, und für $x \geq \texttt{succ}(1.0)$ erhält man die gleiche Fehlerschranke wie für $x \leq \texttt{pred}(1.0)$. Die automatische Berechnung der Fehlerschranke $\varepsilon(f)$ erfolgt mit dem Programm `book_expl27.cpp`, in dem die relative Fehlerschranke $\varepsilon(\ln) = 2.9398 \cdot 10^{-16}$ der C-XSC Logarithmus-Funktion benutzt wird, vgl. Anhang C.

```
// Programm:  book_expl27.cpp
#include "abs_relh.hpp" // Wegen rel_mulh2()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <imath.hpp>    // Wegen Z = ln(Xi);
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval LN_Xi,R,T;
    real del,eps,epsLN;   // epsLN: Fehlerschranke von ln()
    abs_mulh1(Xi,delx,Xi,delx,R,del);      // absl. Fehler bzgl. x*x
    LN_Xi = ln(Xi);
    string("[2.9398e-16,2.9398e-16]") >> T;
    epsLN = Sup(T);       // epsLN:  relativer Fehler der ln-Funktion
    rel_mulh2(R,del,LN_Xi,epsLN,T,eps);  // rel. Fehler:  (x*x)*ln(x)
    return eps;  // Rückgabe einer Schranke eps des relativen Fehlers.
}

int main()
{
    interval X1 = interval(1e-20,pred(1.0));
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X1,delx,diam,bnd);
    cout << RndUp << endl
         << "   **   Relat. Fehlerschranke = " << bnd << endl << endl;
}
```

Nach dem Start des Programms `book_expl27` erhält man die Bildschirmausgabe:
`**   Relat. Fehlerschranke = 7.380781E-016`, d.h. $\varepsilon(f) = 7.380781 \cdot 10^{-16}$.

Nach dem Start des Programms `Min-Incl` und der Dateneingabe

```
    ****************************************
    *   Minimum point approximation xs = ?   *
    ****************************************
0.8
    ************************************************
    *   Relative error bound of f(x),  epsf = ?    *
    ************************************************
7.380781e-16
```

erhält man die Bildschirmausgabe:

```
    **   Please wait some moments ....

    **   x0    is included in: [0.606530640670335,0.606530678743994]
    **   f(x0) is included in: [-0.183939720585723,-0.183939720585720]
```

Das C-XSC Programm `Min-Incl` liefert also mit dem Startwert `xs` $= 0.8$ und $\varepsilon(f) = 7.380781 \cdot 10^{-16}$ die folgenden garantierten Einschließungen für die Minimumstelle x_0 und für das lokale Minimum $f(x_0)$:

$$x_1 = 6.06530640670335 \cdot 10^{-1} < x_0 = \frac{1}{\sqrt{e}} < 6.06530678743994 \cdot 10^{-1} = x_2,$$

$$-1.83939720585723 \cdot 10^{-1} < f(x_0) = \frac{-1}{2e} < -1.83939720585720 \cdot 10^{-1};$$

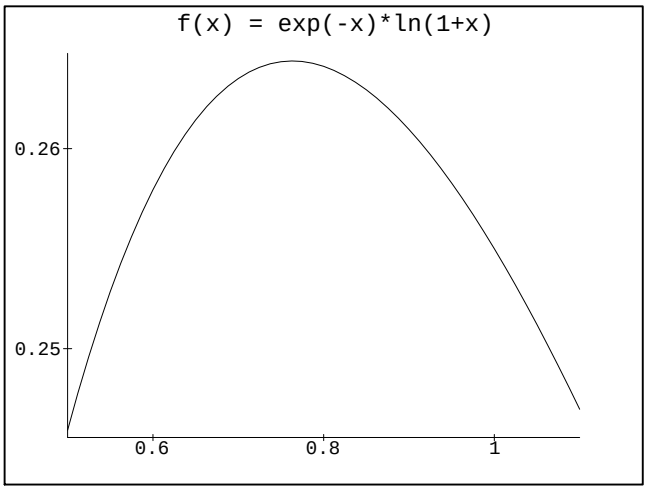
Die letzte Einschließung enthält nicht nur $f(x_0)$ sondern sogar $f([x_1, x_2])$. Beachten Sie, dass der Algorithmus in unserem Beispiel die gesuchten Einschließungen findet, obwohl f nicht im ganzen Definitionsbereich konvex ist!

10.4.2 $f(x) = e^{-x} \cdot \ln(1+x),\ x > -1$

Die zu untersuchende Funktion f ist definiert durch:

$$f(x) = \mathrm{e}^{-x} \cdot \ln(1+x), \quad x > -1;$$

und besitzt nur eine lokale Maximumstelle x_0, die zusammen mit dem Maximum $f(x_0)$ einzuschließen ist. Im Gegensatz zum vorhergehenden Beispiel kann jetzt die Maximumstelle x_0 nicht in geschlossener Form dargestellt werden, da $f'(x) = 0$ mit der folgenden Gleichung $(1+x) \cdot \ln(1+x) = 1$ äquivalent ist.



Berechnung der Fehlerschranke $\varepsilon(f)$:

Benutzt werden die Fehlerschranken der C-XSC Funktionen lnp1x(x), exp(x):

$$\varepsilon(\text{lnp1x}) = 2.5082 \cdot 10^{-16}, \qquad \varepsilon(\text{exp}) = 2.357963 \cdot 10^{-16}$$

Mit dem folgenden Programm `book_expl28.cpp` kann die relative Fehlerschranke $\varepsilon(f)$ der Funktion $f(x)$ automatisch berechnet werden.

```
// Programm:  book_expl28.cpp
#include "abs_relh.hpp" // Wegen rel_mulh2()
#include "bnd_util.hpp" // Wegen Max_bnd_Xi()
#include <imath.hpp>    // Wegen Z = ln(Xi);
#include <iostream>     // Wegen cout

using namespace cxsc;
using namespace std;

real T_x(const interval& Xi, const real& delx)
{
    interval LNP1_Xi,EXP_Xi,T;   real eps,epsLNP1,epsEXP;
    LNP1_Xi = lnp1(Xi);
    string("[2.5082e-16,2.5082e-16]") >> T;
    epsLNP1 = Sup(T);  // epsLNp1:  rel. Fehler von lnp1(x)
    EXP_Xi = exp(-Xi);
    string("[2.357963e-16,2.357963e-16]") >> T;
```

```
    epsEXP = Sup(T);
    rel_mulh4(EXP_Xi,epsEXP,LNP1_Xi,epsLNP1,T,eps);
    return eps;  // Rückgabe einer Schranke eps des relativen Fehlers.
} // T_x

int main()
{
    interval X1;
    string("[0.001,708.39]") >> X1;
    real bnd, diam = 1e-5, delx=0;
    Max_bnd_Xi(T_x,X1,delx,diam,bnd);
    cout << RndUp << endl
         << "   **   Relat. Fehlerschranke = " << bnd << endl << endl;
}
```

Nach dem Start des Programms `book_expl28` erhält man die Bildschirmausgabe: `** Relat. Fehlerschranke = 7.086610E-016`, d.h. $\varepsilon(f) = 7.086610 \cdot 10^{-16}$. Beachten Sie bitte, dass bei der obigen Deklaration

```
string("[0.001,708.39]") >> X1;
```

eine größere Obergrenze, d.h. also z.B. 708.4, einen unendlichen relativen Fehler verursacht, da dann die Werte der Exponentialfunktion in den denormalisierten Bereich fallen und dann durch $[0, \texttt{MinReal}]$ eingeschlossen werden. Mit diesen Einschließungen kann ein relativer Fehler wegen auftretender Divisionen durch Null jedoch nicht bestimmt werden! Für unsere Bedürfnisse sind so große Intervallobergrenzen jedoch ohne Bedeutung, da die relative Fehlerschranke wegen $x_0 \approx 0.7$ nur etwa im Bereich $[0.1, 2]$ benutzt wird.

Nachdem $\varepsilon(f) = 7.086610 \cdot 10^{-16}$ bekannt ist, kommen wir jetzt zur Berechnung garantierter Einschließungen von x_0 und $f(x_0)$. Da das Programm `Min-Incl` nur ein **Minimum** einschließen kann, muss im Quelltext `Min-Incl.cpp` in der C-XSC Funktion `real f(const real& x)` der Faktor (-1) berücksichtigt werden:

```
real f(const real& x)
{   // Definition of the convex function f(x)
    real y;
    y = -exp(-x)*lnp1(x);
    return y;
}
```

Mit dem oben berechneten Wert für $\varepsilon(f) = 7.086610 \cdot 10^{-16}$ und dem Startwert $x_S = 1$ liefert `Min-Incl` die folgende Bildschirmausgabe:

```
**   Please wait some moments ....

**   x0    is included in: [0.763222782539589,0.763222883743809]
**   f(x0) is included in: [-0.264380447349651,-0.264380447349633]
```

Das C-XSC Programm `Min-Incl` liefert also mit dem Startwert `xs` $= 1.0$ und $\varepsilon(f) = 7.086610 \cdot 10^{-16}$ die folgenden garantierten Einschließungen für die Maximumstelle x_0 und für das lokale Maximum $f(x_0)$:

$$x_1 = 0.763222782539589 < x_0 < 0.763222883743809 = x_2,$$

$$0.264380447349633 < f(x_0) < 0.264380447349651;$$

Die letzte Einschließung entsteht dabei aus der entsprechenden Bildschirmausgabe des Programms natürlich nur durch Multiplikation mit dem Faktor (-1) und enthält nicht nur das relative Maximum $f(x_0)$ sondern sogar $f([x_1, x_2])$. Beachten Sie, dass der Algorithmus in unserem Beispiel die gesuchten Einschließungen findet, obwohl $-f(x)$ nicht im ganzen Definitionsbereich konvex ist!

Anhang A

Das Programm Approx-Error

```
/*
**  CXSC is a C++ library for eXtended Scientific Computing (V 2.0_beta3)
**
**  Copyright (C) 1990-2000 Institut fuer Angewandte Mathematik,
**                          Universitaet Karlsruhe, Germany
**            (C) 2000-2003 Wiss. Rechnen/Softwaretechnologie
**                          Universitaet Wuppertal, Germany
**
**  This program 'Approx-Error' is free software; you can redistribute it
**  and/or modify it under the terms of the GNU Library General Public
**  License as published by the Free Software Foundation; either
**  version 2 of the License, or (at your option) any later version.
**
**  This program 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.  See the GNU
**  Library General Public License for more details.
**
**  You should have received a copy of the GNU Library General Public
**  License along with this library; if not, write to the Free
**  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
*************************************************************************
**  The program     'Approx-Error'     calculates an upper bound of the   **
**  absolute and relative approximation error by rational approximation.  **
**  f(x) is approximated by the fraction      P_M(x-x1)/Q_N(x-x1)         **
**  Kraemer|Blomquist, 17.01.05;                                          **
*************************************************************************
*/
```

```
#include <iostream>      // For cout, cerr, ...
#include "litaylor.hpp"  // For l_itaylor type
#include <imath.hpp>     // For power(...)

using namespace std;
using namespace cxsc;
using namespace taylor;

const int Z1 = 1;   // Number of subintervals for the remainder term
const int Z2 = 50;  // Each subinterval will again be subdivided by
                    // Z2 new subintervals. The expansion point for
                    // all these Z2 subintervalls is x0.

// ***************************************************************************
// *****   The function f(x) to be approximated must be defined here:   ****
// ***************************************************************************
l_itaylor STD(const l_itaylor& x)
{   // Definition of the function to be approximated
    // by a rational function
    l_itaylor t;
    t = exp(x);
    return t;
}
// ***************************************************************************

l_itaylor TAYLOR_STD(const l_interval& x0, int ord)
// Calculating the staggered Taylor coefficients of the function defined
// in STD(...) for the expansion interval x0 up to order ord.
// Diam(x0) should be as small as possible to avoid overestimations!
{
    l_itaylor arg = var_l_itaylor(ord, x0);
    return STD(arg);
}

interval POLY_HORN(const interval& arg, const ivector& coef)
// POLY_HORN evaluates a polynomial of degree Ub(coef) for the interval
// argement arg. The polynomial coefficients are given by the ivector coef
// with components of type interval and indices from 0,1,... up to Ub(coef)
{
    interval t;
    t = coef[Ub(coef)]; // polynom. coefficient with maximal index Ub(coef)
    for (int k=Ub(coef)-1; k>=0; k--) // Horner's scheme
        t = t*arg + coef[k];
    return t;
}

void REST_TEIL(const interval& x, int K, real& RN, real& fx)
// Output values:
// RN: Upper bound of all absolute values of the (K+1)-th Taylor
```

```
//     coefficient of STD(...) for all arguments of the subinterval x.
// fx: Lower bound of all absolute function values of STD(...) for all
//     arguments of the subinterval x.
// Input values:
// x:  A subinterval of type interval;
// K:  The degree of Taylor expansion of STD(...).
{
    l_interval x_li;
    l_itaylor t;
    interval u;

    x_li = x;
    t = TAYLOR_STD(x_li,K+1);

    u = get_j_coef(t, K+1); // u: Inclusion of the (K+1)-th Taylor coeffic.
    RN = Sup(abs(u));

    u = get_j_derive(t,0);
    fx = Inf(abs(u));
} // REST_TEIL

void TAU(const interval& x, int K, int Nr, real& tau_abs, real& tau_rel)
// Output:
// tau_abs: Upper bound of the absolute remainder values over interval x
//          for calculating the absolute error.
// tau_rel: Upper bound of the absolute remainder values over interval x
//          for calculating the relative error.
// Input:
// x :      The considered interval; its mid-point is the
//          Taylor expansion point x0.
// K :      The degree of Taylor expansion
// Nr:      The number of subintervals to avoide overestimations by
//          calculating the Taylor-coefficients and the Lower bound
//          of the absolute values of f(x); Nr = 1,2,3, ...
{
    real delta,c1,c2;
    interval xt; // xt is one of the Nr subintervals

    tau_abs = 0;  tau_rel = MaxReal;
    delta = diam(x) / Nr;
    Sup(xt) = Inf(x);

    for (int k=1; k<=Nr; k++)
    {
        Inf(xt) = Sup(xt);

        Sup(xt) = (k==Nr) ? Sup(x) : Inf(xt) + delta;

        REST_TEIL(xt,K,c1,c2);
```

```
        if (c1>tau_abs) tau_abs = c1;
        if (c2<tau_rel) tau_rel = c2;
    }
    c1 = divu(diam(x),2);
    xt = power(interval(c1),K+1);

    tau_abs = mulu(tau_abs,Sup(xt));
    tau_rel = (tau_rel==0) ? MaxReal : divu(tau_abs,tau_rel);

} // TAU

void ERROR_TEIL(const interval& x,  // Considered subinterval
                const interval& x0, // Expansion interval with diam=0 for
                                    // the Taylor expansion of f(x);
                                    // (x-x0) is the polynomial argument.
                const ivector& z,   // Polynomial coefficients of the
                                    // numerator. These coefficients are
                                    // calculated in staggered Arithmetic
                                    // with regard to x0.
                const ivector& b,   // Polynomial coefficients of the
                                    // denominator. These coefficients are
                                    // calculated in staggered Arithmetic
                                    // with regard to x0.
                const ivector& sN,  // K+1 coefficients of the Taylor
                                    // polynomial for the approximation of
                                    // f(x) with regard to x0.
                const real& tau_abs,// Upper bound for the absolute values
                                    // of the remainder term with regard to x.
// Output:
                real& abs_,         // Upper bound of the absolute
                                    // approximation error of the first
                                    // error term.
                real& rel_)         // Upper bound of the relative
                                    // approximation error of the first
                                    // error term.
{
    interval Zae,Ne1,Ne2,c;
    c = interval(-tau_abs,+tau_abs);
    Zae = POLY_HORN(x-x0,z);
    Ne1 = POLY_HORN(x-x0,b);
    Ne2 = Ne1 * (POLY_HORN(x-x0,sN)+c);
    if(0 <= Ne1) {abs_ = MaxReal; rel_ = MaxReal;}
    else {
        abs_ = Sup(abs(Zae/Ne1));
        rel_ = (0 <= Ne2) ? MaxReal : Sup(abs(Zae/Ne2));
    }
} // ERROR_TEIL
```

```
void SUMMAND1(
//  Input:
    const interval& x,   // considered subinterval
    const interval& x0,  // Expansion interval with diam(x0)=0 for the
                         // Taylor expansion of f(x)
                         // (x-x0) is the polynomial argument.
    const ivector& z,    // Polynomial coefficients of the numerator.
                         // These coefficients are calculated in staggered
                         // Arithmetic with regard to x0.
    const ivector& b,    // Polynomial coefficients of the denominator.
                         // These coefficients are calculated in staggered
                         // Arithmetic with regard to x0.
    const ivector& sN,   // K+1 coefficients of the Taylor polynomial for
                         // the approximation of f(x) with regard to x0.
          int Nr,        // Number of subintervals in which x will be
                         // subdivided.
    const real& tau_abs, // Upper bound of the absolute values of the
                         // remainder term with regard to the subinterval x
// Output:
          real& abs_e,   // Upper bound of the absolute approximation error
                         // with regard to the first error term.
          real& rel_e)   // Upper bound of the relative approximation error
                         // with regard to the first error term.
{
    real delta,c1,c2;
    interval xt;  // one of the Nr subintervals

    abs_e = 0;   rel_e = 0;
    delta = diam(x) / Nr;
    Sup(xt) = Inf(x);
    for (int k=1; k<=Nr; k++) {
        Inf(xt) = Sup(xt);
        Sup(xt) = (k==Nr) ? Sup(x) : Inf(xt) + delta;
        ERROR_TEIL(xt,x0,z,b,sN,tau_abs,c1,c2);
        if (c1>abs_e) abs_e = c1;
        if (c2>rel_e) rel_e = c2;
    }
} // SUMMAND1

l_itaylor POLY_ZAEHLER(
    const l_itaylor& x,
    const l_interval& x1, // expansion point x1.
    const l_ivector& ci)  // Staggered inclusions of the numerator
                          // polynomial coefficients with regard to
                          // the expansion point x1.
{
    l_itaylor t = const_l_itaylor(get_order(x),ci[Ub(ci)]);
    for (int k=Ub(ci)-1; k>=0; k--) {
        t = t * (x-x1) + ci[k];   // Horner's scheme
```

```
    }
    return t;
} // POLY_ZAEHLER

l_itaylor TRANS_Z(
    const l_interval x0,  // Evaluation point
          int n,
    const l_interval& x1, // expansion point x1
    const l_ivector& ci)
{
    l_itaylor arg = var_l_itaylor(n,x0);
    return POLY_ZAEHLER(arg,x1,ci);
} // TRANS_Z

l_itaylor POLY_NENNER(
    const l_itaylor& x,
    const l_interval& x1, // expansion point x1.
    const l_ivector& di)  // Staggered inclusions of the numerator
                          // polynomial coefficients with regard to
                          // the expansion point x1.
{
    l_itaylor t = const_l_itaylor(get_order(x),di[Ub(di)]);
    for (int k=Ub(di)-1; k>=0; k--) {
        t = t * (x-x1) + di[k];   // Horner's scheme
    }
    return t;
}

l_itaylor TRANS_N(
    const l_interval x0,  // Evaluation point
          int n,
    const l_interval& x1, // expansion point x1
    const l_ivector& di)
{
    l_itaylor arg = var_l_itaylor(n,x0);
    return POLY_NENNER(arg,x1,di);
} // TRANS_N

void Approx_Fehler(
// This function calculates the abs. and rel. error bounds for one
// of the Z subintervals of ab.
// Input:
    const interval& intv, // The considered subinterval
          int K,          // Degree of Taylor polynomial (with mid-point
                          // of intv) for the approximation of f(x)
          int P,          // Pecision parameter for litaylor
          int M,          // Degree of the numerator polynomial of the
```

```
                          // rational approximation
          int N,          // Degree of the denominator polynomial of the
                          // rational approximation
    const l_ivector& ci,  // Staggered inclusions of the numerator
                          // polynomial coefficients with regard to
                          // the expansion point x1.
    const l_ivector& di,  // Staggered inclusions of the denominator
                          // polynomial coefficients with regard to
                          // the expansion point x1.
    const l_interval& x1, // The coefficients ci,di (for example
                          // calculated with Mathematica) are related to
                          // the expansion point x1.
// Output:
          real& abs_err,  // Upper bound of the absolute approx. error
          real& rel_err)  // Upper bound of the absolute approx. error
{
    stagprec = P;
    real tau_abs,tau_rel;
    int Bh;
    l_interval x0  = l_interval(mid(intv));
    l_ivector  a0  = l_ivector(0,M);
    l_ivector  b0  = l_ivector(0,N);
    l_ivector  s0  = l_ivector(0,K+1);
    l_ivector  z0  = l_ivector(0,K+N);
    l_itaylor  tz;
    l_itaylor  tn;
    l_itaylor  t0;
    ivector    sN  = ivector(0,K);
    ivector    z   = ivector(0,K+N);
    ivector    b   = ivector(0,N);
    interval x_Null;

    if (x0==x1) { a0 = ci;  b0 = di;}
    else {
        tz = TRANS_Z(x0,Ub(a0),x1,ci);
        for (int j=0; j<=Ub(a0); j++)
            a0[j] = get_j_coef(tz,j);
        tn = TRANS_N(x0,Ub(b0),x1,di);
        for (int j=0; j<=Ub(b0); j++)
            b0[j] = get_j_coef(tn,j);
    }
    t0 = TAYLOR_STD(x0,Ub(s0));
    for (int j=0; j<=Ub(s0); j++)
        s0[j] = get_j_coef(t0,j);
    for (int j=0; j<=Ub(sN); j++)
        sN[j] = s0[j];
// Now the polynomial coefficients z[j] of the numerator for calculating
// the absolute or relative approximation error are computed:
```

```
// Polynomial multiplication:

    for (int k=0; k<=Ub(b0); k++) {
        z0[k] = 0;
        for (int j=0; j<=k; j++)
            z0[k] = z0[k] + b0[k-j]*s0[j];
    }
    for (int k=Ub(b0)+1; k<=Ub(s0)-1; k++) {
        z0[k] = 0;
        for (int j=0; j<=Ub(b0); j++)
            z0[k] = z0[k] + b0[Ub(b0)-j]*s0[k-Ub(b0)+j];
    }
    Bh = Ub(s0)-1+Ub(b0);
    for (int k=Ub(s0); k<=Bh; k++) {
        z0[k] = 0;
        for (int j=0; j<=Bh-k; j++)
            z0[k] = z0[k] + b0[Ub(b0)-j]*s0[k-Ub(b0)+j];
    }  // End of polynomial multiplication

// Polynomial subtraction:
    for (int k=0; k<=Ub(a0); k++)
        z0[k] = z0[k] - a0[k];
    for (int k=0; k<=Ub(z); k++)
        z[k] = z0[k];

    TAU(intv,K,Z1,tau_abs,tau_rel);

    for (int k=0; k<=Ub(b); k++)
        b[k] = b0[k];
    x_Null = x0;

    SUMMAND1(intv,x_Null,z,b,sN,Z2,tau_abs,abs_err,rel_err);

    if (abs_err==MaxReal || tau_abs==MaxReal) abs_err = MaxReal;
    else abs_err = addu(abs_err,tau_abs);

    if (rel_err==MaxReal || tau_rel==MaxReal) rel_err = MaxReal;
    else rel_err = addu(rel_err,tau_rel);
} // Approx_Fehler

void Haupt(const interval& ab,  // ab: Approximation interval
           int Z,               // Z:  Number of subintervals
           int K,               // K:  Degree of Taylor polynomial for f(x)
           int M,               // M:  Degree of numerator polynomial p
           int N,               // N:  Degree of denominator polynomial q
           const real& x1,      // x1: Expansion point of rat. approximation
           const rvector& p,    // Polynomial coefficients of the numerator
           const rvector& q,    // Polynomial coefficients of the denominator
                 real& abs_err, // abs_err: Absolute error bound
```

```
                real& rel_err) // rel_err: Relative error bound
{
    int P;        // P: actual stagprec value
    interval xt;  // xt: One of the Z subintervals
    real delta,   // Step size
         ca,cr;   // Absolute and relative error bound
    l_interval x1l = l_interval(x1);
    l_ivector ci = l_ivector(0,M);
    l_ivector di = l_ivector(0,N);

    P = stagprec;

    for (int k=0; k<=M; k++)
        ci[k] = l_interval(p[k]);
    for (int k=0; k<=N; k++)
        di[k] = l_interval(q[k]);

    abs_err = 0;  rel_err = 0;
    delta = diam(ab) / Z;
    Sup(xt) = Inf(ab);
    for (int k=1; k<=Z; k++)
    {
        Inf(xt) = Sup(xt);
        if (k==Z) Sup(xt) = Sup(ab);
        else Sup(xt) = Inf(xt) + delta;

        Approx_Fehler(xt,K,P,M,N,ci,di,x1l,ca,cr);

        if (ca>abs_err) abs_err = ca;
        if (cr>rel_err) rel_err = cr;
    }
} // Haupt()

int main()
{
    string str = "**********************************";
    string strleer = "    ";
    char Zeichen = char(247);
    int M,   // Polynomial degree of p_M(x-x1)
        N,   // Polynomial degree of q_N(x-x1)
        Z,   // Number of subintervals in the approximation interval ab
        K,   // Degree of Taylor expansion of f(x)
        max, // max = max{M,N}
        P;   // stagprec value
    double dbl; // Will be only used by function fscanf(...)
    real x1; // Expansion point for the given rational approximation
             // f(x) approx p_M(x-x1) / q_N(x-x1);
             // p_M(x-x1) := p[0] + ... + p[M]*(x-x1)^M;
             // q_N(x-x1) := q[0] + ... + q[N]*(x-x1)^N;
```

```
          // The polynomial degrees M,N, the expansion point x1 and
          // the coefficients p[i],q[j] are stored in this sequence
          // in decimal form in the file APPR-Dat.txt which must be
          // located in the same directory where this program will be
          // started.
real abs_err, // Absolute error bound
     rel_err; // Relative error bound

interval ab;  // ab: Approximation interval

FILE *fp;

if ( (fp = fopen("APPR-Dat.txt", "rb")) == 0)
{
    cerr << "Error by opening file" << endl;
    exit(1);
}
cout << endl;
cout << strleer+str+str << endl;
cout << strleer+"*   "+
     "                                                                  *"
     << endl;
cout << strleer+"*   "+
     "This program calculates guaranted absolute and relative error    *"
     << endl;
cout << strleer+"*   "+
     "bounds for the rational approximation                             *"
     << endl;
cout << strleer+"*   "+
     "                 f(x) "+char(247)+" P_M(x-x1) / Q_N(x-x1).       *"
     << endl;
cout << strleer+"*   "+
     "f(x), defined by standard functions, must be implemented in      *"
     << endl;
cout << strleer+"*   "+
     "the internal function STD(...).                                   *"
     << endl;
cout << strleer+"*   "+
     "The polynomial degrees M>0, N>=0 of the polynomials P_M(x-x1),    *"
     << endl;
cout << strleer+"*   "+
     "Q_N(x-x1), the expansion point x1, which must be representable    *"
     << endl;
cout << strleer+"*   "+
     "in the IEEE-double format, and the polynomial coefficients        *"
     << endl;
cout << strleer+"*   "+
     "p[j],q[j], j=0,1,...,M|N, must be written in this sequence in     *"
     << endl;
```

```
cout << strleer+"*   "+
     "decimal form to the text-file 'APPR-Dat.txt', which must be      *"
     << endl;
cout << strleer+"*   "+
     "stored in the same directory, where this program is running.     *"
     << endl;
cout << strleer+"*   "+
     "By reading the polynomial coefficients with at least 18 decimal  *"
     << endl;
cout << strleer+"*   "+
     "digits from the text file, these coefficients are rounded to     *"
     << endl;
cout << strleer+"*   "+
     "the next IEEE-numbers p[j],q[j], which will then be treated as   *"
     << endl;
cout << strleer+"*   "+
     "the exact polynomial coefficients!    The text file content is   *"
     << endl;
cout << strleer+"*   "+
     "listed below; to continue press any key!                         *"
     << endl;
cout << strleer+"*   "+
     "                                                                 *"
     << endl;
cout << strleer+str+str << endl;

fscanf(fp," %d %d %lf ", &M,&N,&dbl);
cout << strleer+"*   "+"M = " << M << endl;
cout << strleer+"*   "+"N = " << N << endl;

max = M;
if (N>max) max = N;

rvector p(0,M);  // Coefficients of the polynomial p_M(x-x1) of order M
rvector q(0,N);  // Coefficients of the polynomial q_N(x-x1) of order N

x1 = dbl;  // x1: Expansion point of the given rational approximation
cout << SetPrecision(18,18) << Scientific
     << strleer+"*   "+"Expansion point x1 = " << x1 << endl;

for (int k=0; k<=M; k++)
{
    fscanf(fp," %lf ", &dbl);  // Rounding the next decimal string in
    p[k] = dbl;                // the file APPR-Dat.txt to the nearest
                               // IEEE-real number p[k];
    cout << strleer+"*   "+"p[" << k << "] = " << p[k] << endl;
}
for (int k=0; k<=N; k++)
{
```

```
        fscanf(fp," %lf ", &dbl);  // Rounding the next decimal string in
        q[k] = dbl;                // the file APPR-Dat.txt to the nearest
                                   // IEEE-real number q[k];
        cout << strleer+"*   "+"q[" << k << "] = " << q[k] << endl;
    }

    fclose(fp); // closing the Text-File APPR-Dat.txt in the current directory
    do {
    cout << "    ************************************** " << endl;
    cout << "    *   Approximation intervall [a,b] = ?   * " << endl;
    cout << "    ************************************** " << endl;
    cin >> ab;
    } while (Inf(ab)>Sup(ab));
    cout << "    ********************************************** " << endl;
    cout << "    *   Z: Number of subintervals in [a,b];  Z = ?   * " << endl;
    cout << "    ********************************************** " << endl;
    cin >> Z;
    if (Z<1) Z = 1;
    cout << "    ****************************************** " << endl;
    cout << "    *   f(x): Taylor expansion of degree K = ?   * " << endl;
    cout << "    ****************************************** " << endl;
    cin >> K;
    if (K<=max) K = max + 1;
    cout << "    Used value K = " << K << endl;
    cout << "    ************************************* " << endl;
    cout << "    *   Multiprecision parameter P = ?   * " << endl;
    cout << "    ************************************* " << endl;
    cin >> P;
    if (P<2) P = 2;
    stagprec = P;
    cout << "    Used stagprec value P = " << P << endl << endl;
    cout << "    Please wait some moments ...." << endl << endl;
    Haupt(ab,Z,K,M,N,x1,p,q,abs_err,rel_err);
    cout << SetPrecision(10,10);
    cout << RndUp << "    *   Error bound for the absolute error:  "
         << abs_err << endl;
    cout << RndUp << "    *   Error bound for the relative error:  "
         << rel_err << endl << endl;
} // main
```

Anhang B

Das Programm Min-Incl

```
/*
**  CXSC is a C++ library for eXtended Scientific Computing (V 2.0_beta3)
**
**  Copyright (C) 1990-2000 Institut fuer Angewandte Mathematik,
**                          Universitaet Karlsruhe, Germany
**            (C) 2000-2003 Wiss. Rechnen/Softwaretechnologie
**                          Universitaet Wuppertal, Germany
**
**  This program 'Min-Incl' is free software; you can redistribute it
**  and/or modify it under the terms of the GNU Library General Public
**  License as published by the Free Software Foundation; either
**  version 2 of the License, or (at your option) any later version.
**
**  This program 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.  See the GNU
**  Library General Public License for more details.
**
**  You should have received a copy of the GNU Library General Public
**  License along with this library; if not, write to the Free
**  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
*/

/*
*************************************************************************
**  The program    'Min-Incl'    calculates an inclusion [x1,x2] of the   **
**  the minimum point x0 and an inclusion [y1,y2]  of the minimum value   **
**  f(x0) of a function  f ,which is convex and has a relative minimum.   **
**  Furthermore [y1,y2] is an inclusion of f([x1,x2]).                    **
**  Kraemer|Blomquist, 25.01.05;                                          **
*************************************************************************
*/
```

```
#include <iostream>      // For cout, cerr, ...
#include <interval.hpp>
#include <rmath.hpp>

using namespace std;
using namespace cxsc;

// ************************************************************************
// *****     The real convex function   f(x)  must be defined here:     ****
// ************************************************************************

real f(const real& x)
{   // Definition of the convex function f(x)
    real y;
    y = (x>0) ? x*x*ln(x) : x*x;
    return y;
}
// ************************************************************************
// ************************************************************************

void SCHRANKE(const real& x,    // function argument
              const real& eps,  // Relative error bound of the function
              real& fu,         // Lower bound of the exact value f(x)
              real& fo)         // Upper bound of the exact value f(x)
{
    real amin, aplus, fw;
    int k;
    amin  = subd(1.0,eps);
    aplus = addu(1.0,eps);
    fw = f(x);

    if (fw==0) k=0;
    if (fw>0)  k=1;
    if (fw<0)  k=2;
    switch(k)
    {
        case 0:
            fu=0; fo=0;
            break;
        case 1:
            fu = divd(fw,aplus);
            fo = divu(fw,amin);
            break;
        case 2:
            fu = divd(fw,amin);
            fo = divu(fw,aplus);
            break;
    }
} // SCHRANKE
```

```
void X_interval(const real& eps, // Relative error bound of f(x)
                const real& ZP,  // Factor determing the step size
                real& xs,        // For a given approximation xs of x0
                                 // an inclusion [xa,xb] is calculated
                real& xa,
                real& xb)        // xs=0.5*(xa+xb) is a new approximation
/*  For a first approximation xs of x0 an inclusion interval [xa,xb]   */
/*  for the minimum point xo is calculated.       The returned value   */
/*  xs := 0.5*(xa+xb) is then an improved new approximation of x0.     */
{
    real OS,fu,fo,x,delta,xalt;
    int n;
    SCHRANKE(xs,eps,fu,OS);      // fu < f(xs) < OS;
    x = xs;  delta = abs(x*ZP);
    n = 0;
    do {
        xalt = x;
        x = subd(x,delta);  // running to the left side
        SCHRANKE(x,eps,fu,fo);
        n++;
        if (n==10) {n = 0;  delta *= 1.5;}
    } while (fu<OS);
    delta /= 200;  // refinement of the left value xalt
    while (xalt == xalt-delta) delta *= 1.5;

    do {
        xalt = subd(xalt,delta);
        SCHRANKE(xalt,eps,fu,fo);
        delta *= 1.5;
    } while (fu<=OS);
    xa = xalt; // xa < x0
    x = xs;  delta = abs(x*ZP);  n = 0;
    do {
        xalt = x;  x = addu(x,delta);  // running to the right side
        SCHRANKE(x,eps,fu,fo);
        n++;
        if (n==10) {n = 0; delta *= 1.5;}
    } while (fu<OS);
    delta /= 200;  // refinement of the right value xalt
    while (xalt == xalt+delta) delta *= 1.5;

    do {
        xalt = addu(xalt,delta);
        SCHRANKE(xalt,eps,fu,fo);
        delta *= 1.5;
    } while (fu<=OS);
    xb = xalt;  // xa < x0 < xb
    xs = (xa+xb) / 2;  // xs: A new approximation for x0
} // X_interval
```

```
void X_itera(const real& eps,
             real& ZP,
             real& xs,
             real& xa,
             real& xb)
/*  xs: Approximation of the exact minimum point x0.            */
/*  ZP = 10^(-2); |xs*ZP| is the initial step size.             */
/*  [xa,xb] is an inclusion of x0; several iteration are done.  */
{
    real x1,x2,x1alt,x2alt,lalt,lneu;
    x1 = -1e306;  x2 = -x1;  lneu = x2 - x1;
    do {
        x1alt = x1;  x2alt = x2;  lalt = lneu;
        X_interval(eps,ZP,xs,x1,x2);
        ZP /= 16;  lneu = x2 - x1;
    } while (lneu<lalt);
    xa = x1alt;   xb = x2alt;
} // X_itera

void MIN_EINSCHL(const real& eps,
                 real& ZP,
                 real& xs,
                 real& x1,
                 real& x2,
                 real& y1,
                 real& y2)
{
    real xa,xb,deltax,x,y,y1l,y2l,m1,m2,m,m1alt,b1,b2,c1,c2;
    int n;

    X_itera(eps,ZP,xs,xa,xb);  // xa < x0 < xb
    // Calculation of the right secant equation:  y=m1*x+b1, m1>0;
    deltax = (xb-xa) / 100;  m1 = 1e306;
    SCHRANKE(xb,eps,y1l,y);

    do {
        m1alt = m1;
        deltax *= 1.1;
        x = addu(xb,deltax); // running to the right to decrease
        // the secant slope until this slope will increase again.
        SCHRANKE(x,eps,y,y2l);
        c1 = subu(y2l,y1l);
        c2 = subd(x,xb);
        m1 = divu(c1,c2);    // m1: positive secant slope
    } while (m1<=m1alt);
    deltax /= 1.21;  // Going two step backwards; starting from
    // this point with a smaller step size to the right for getting
    // a more smaller secant slope.
    m = m1alt;   m1 = 1e306;
```

```
    do {
        m1alt = m1;
        deltax *= 1.001;
        x = addu(xb,deltax); // The secant slope should be as small
                             // as possible!
        SCHRANKE(x,eps,y,y2l);
        c1 = subu(y2l,y1l);
        c2 = subd(x,xb);
        m1 = divu(c1,c2);
    } while (m1<m1alt);
    m1 = m1alt;
    if (m1>m) m1 = m; // m1: smallest positive secant slope
    if (xb>0) { c1 = mulu(m1,xb); b1 = subd(y1l,c1);}
    else { c1 = muld(m1,xb); b1 = subd(y1l,c1);} // y=m1*x+b1;

// Calculaiton of the left secant equation:
    deltax = (xb-xa) / 100;  m2 = -1e306;
    SCHRANKE(xa,eps,y1l,y);
    do {
        m1alt = m2;
        deltax *= 1.1;
        x = subd(xa,deltax);
        SCHRANKE(x,eps,y,y2l);
        c1 = subd(y1l,y2l);
        c2 = subd(xa,x);
        m2 = divd(c1,c2);
    } while (m2>=m1alt);
    deltax /= 1.21;   m = m1alt;
    m2 = -1e306;
    do {
        m1alt = m2;   deltax *= 1.001;
        x = subd(xa,deltax);
        SCHRANKE(x,eps,y,y2l);
        c1 = subd(y1l,y2l);
        c2 = subd(xa,x);
        m2 = divd(c1,c2);
    } while (m2>m1alt);
    m2 = m1alt;
    if (m2<m) m2 = m;
    if (xa>0) {c1 = muld(m2,xa); b2 = subd(y1l,c1);}
    else {c1 = mulu(m2,xa); b2 = subd(y1l,c1);} // y = m2*x + b2;
// Calculating the point of intersection of the two secants:
    y = subd(b2,b1);
    if (y>0) {c1 = subu(m1,m2);  y = divd(y,c1);}
    else {c1 = subd(m1,m2);  y = divd(y,c1);}
    // y: downrounded x-coordinate of the secant intersection point
    c1  = muld(m1,y);
    y1l = addd(c1,b1);
    // y1l: downrounded y-coordinate of the secant intersection point
```

```
    // it holds:   y1l <= f(x0);
    x1 = xa;
    x2 = xb;  // [x1,x2] = [xa,xb]: Inclusion of the minimum point x0
    SCHRANKE(xa,eps,y,y2l);
    SCHRANKE(xb,eps,y,m1);
    if (m1>y2l) y2l = m1;
    y2 = y2l;
    y1 = y1l; // [y1,y2]: Inclusion of f(x0) and of f([x1,x2]);
} // MIN_EINSCHL

int main()
{
    string str = "**********************************";
    string strleer = "    ";
    real xs, eps, x1, x2, y1, y2, ZP = 1e-2;
    interval xe,ye;
    cout << endl;
    cout << strleer+str+str << endl;
    cout << strleer+"*   "+
         "                                                                  *"
         << endl;
    cout << strleer+"*   "+
         "This program calculates an inclusion of the minimum point  x0    *"
         << endl;
    cout << strleer+"*   "+
         "and an inclusion of the exact value f(x0), if f is convex and    *"
         << endl;
    cout << strleer+"*   "+
         "has a relative minimum at x = x0.                                 *"
         << endl;
    cout << strleer+"*   "+
         "An approximation of x0 and the relative error bound eps(f) of     *"
         << endl;
    cout << strleer+"*   "+
         "the convex funktion f(x) must be known.                           *"
         << endl;
    cout << strleer+"*   "+
         "f(x)  must be implemented in the source code of this program.     *"
         << endl;
    cout << strleer+"*   "+
         "                                                                  *"
         << endl;
    cout << strleer+str+str << endl;

    cout << "    ***************************************** " << endl;
    cout << "    *   Minimum point approximation xs = ?   * " << endl;
    cout << "    ***************************************** " << endl;
    cin >> xs;
```

```
    cout << "    ********************************************** " << endl;
    cout << "    *   Relative error bound of f(x),  epsf = ?    * " << endl;
    cout << "    ********************************************** " << endl;
    cin >> RndUp >> eps;
    cout << endl;
    cout << "    **   Please wait some moments ...." << endl << endl;
    MIN_EINSCHL(eps,ZP,xs,x1,x2,y1,y2);
    xe = interval(x1,x2);  ye = interval(y1,y2);
    cout << SetPrecision(15,15)
         << "    **   x0    is included in: " << xe << endl;

    cout << "    **   f(x0) is included in: " << ye << endl << endl;
} // main
```

Anhang C

Fehlerschranken der Standardfunktionen

Tabelle C.1: Fehlerschranken der Standardfunktionen

Funktion	$\varepsilon(f)$	$1-\varepsilon(f)$	$1+\varepsilon(f)$
$\sqrt{x}$	$2.220447 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
$\sqrt{x+1}-1$	$4.996004 \cdot 10^{-16}$	$\frac{9007199254740984.0}{9007199254740992.0}$	$\frac{4503599627370502.0}{4503599627370496.0}$
$\ln(\sqrt{x^2+y^2})$	$5.160563 \cdot 10^{-16}$	$\frac{9007199254740984.0}{9007199254740992.0}$	$\frac{4503599627370502.0}{4503599627370496.0}$
$\sqrt{x^2-1}$	$2.221305 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
$\sqrt{1-x^2}$	$3.700747 \cdot 10^{-16}$	$\frac{9007199254740985.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
e^x	$2.357962555295842 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
e^x-1	$2.592561649228397 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
e^{-x^2}	$4.618919 \cdot 10^{-16}$	$\frac{9007199254740984.0}{9007199254740992.0}$	$\frac{4503599627370502.0}{4503599627370496.0}$
2^x	$2.350296792932261 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
10^x	$2.418059815583812 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite

Funktion	$\varepsilon(f)$	$1-\varepsilon(f)$	$1+\varepsilon(f)$
$\ln(x)$	$2.9398 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
$\ln(1+x)$	$2.5082 \cdot 10^{-16}$	$\frac{9007199254740986.0}{9007199254740992.0}$	$\frac{4503599627370501.0}{4503599627370496.0}$
$\log_2(x)$	$2.7754 \cdot 10^{-15}$	$\frac{9007199254740964.0}{9007199254740992.0}$	$\frac{4503599627370512.0}{4503599627370496.0}$
$\log_{10}(x)$	$2.7754 \cdot 10^{-15}$	$\frac{9007199254740964.0}{9007199254740992.0}$	$\frac{4503599627370512.0}{4503599627370496.0}$
$\sin(x)$	$1.071713978232866 \cdot 10^{-15}$	$\frac{9007199254740979.0}{9007199254740992.0}$	$\frac{4503599627370504.0}{4503599627370496.0}$
$\cos(x)$	$1.071713978232866 \cdot 10^{-15}$	$\frac{9007199254740979.0}{9007199254740992.0}$	$\frac{4503599627370504.0}{4503599627370496.0}$
$\tan(x)$	$2.97768 \cdot 10^{-15}$	$\frac{9007199254740962.0}{9007199254740992.0}$	$\frac{4503599627370513.0}{4503599627370496.0}$
$\cot(x)$	$2.97768 \cdot 10^{-15}$	$\frac{9007199254740962.0}{9007199254740992.0}$	$\frac{4503599627370513.0}{4503599627370496.0}$
$\arcsin(x)$	$2.148875977690793 \cdot 10^{-15}$	$\frac{9007199254740969.0}{9007199254740992.0}$	$\frac{4503599627370509.0}{4503599627370496.0}$
$\arccos(x)$	$2.148489042525242 \cdot 10^{-15}$	$\frac{9007199254740969.0}{9007199254740992.0}$	$\frac{4503599627370509.0}{4503599627370496.0}$
$\arctan(x)$	$1.358774060669230 \cdot 10^{-15}$	$\frac{9007199254740967.0}{9007199254740992.0}$	$\frac{4503599627370506.0}{4503599627370496.0}$
$\operatorname{arccot}(x)$	$1.802884893838539 \cdot 10^{-15}$	$\frac{9007199254740972.0}{9007199254740992.0}$	$\frac{4503599627370508.0}{4503599627370496.0}$
$\sinh(x)$	$7.093289735801012 \cdot 10^{-16}$	$\frac{9007199254740982.0}{9007199254740992.0}$	$\frac{4503599627370503.0}{4503599627370496.0}$
$\cosh(x)$	$4.581660384746620 \cdot 10^{-16}$	$\frac{9007199254740984.0}{9007199254740992.0}$	$\frac{4503599627370502.0}{4503599627370496.0}$
$\tanh(x)$	$1.054585718561371 \cdot 10^{-15}$	$\frac{9007199254740979.0}{9007199254740992.0}$	$\frac{4503599627370504.0}{4503599627370496.0}$
$\coth(x)$	$8.325226430245611 \cdot 10^{-16}$	$\frac{9007199254740981.0}{9007199254740992.0}$	$\frac{4503599627370503.0}{4503599627370496.0}$
$\operatorname{arsinh}(x)$	$7.2075 \cdot 10^{-16}$	$\frac{9007199254740982.0}{9007199254740992.0}$	$\frac{4503599627370503.0}{4503599627370496.0}$
$\operatorname{arcosh}(x)$	$1.6180 \cdot 10^{-15}$	$\frac{9007199254740974.0}{9007199254740992.0}$	$\frac{4503599627370507.0}{4503599627370496.0}$
$\operatorname{artanh}(x)$	$1.264618646263405 \cdot 10^{-15}$	$\frac{9007199254740977.0}{9007199254740992.0}$	$\frac{4503599627370505.0}{4503599627370496.0}$
$\operatorname{arcoth}(x)$	$1.147880588000001 \cdot 10^{-15}$	$\frac{9007199254740978.0}{9007199254740992.0}$	$\frac{4503599627370505.0}{4503599627370496.0}$

Fortsetzung auf der nächsten Seite

Fortsetzung von vorheriger Seite			
Funktion	$\varepsilon(f)$	$1-\varepsilon(f)$	$1+\varepsilon(f)$
$\operatorname{arcosh}(1+x)$	$7.792706 \cdot 10^{-16}$	$\frac{9007199254740981.0}{9007199254740992.0}$	$\frac{4503599627370503.0}{4503599627370496.0}$

Anmerkungen zur Tabelle C.1:

1. Will man einen Dezimalwert $\varepsilon(f)$ aus Tabelle C.1 in einem Programm korrekt als Oberschranke des relativen Fehlers der entsprechenden Standardfunktion $f(x)$ benutzen, so muss dieser Dezimalwert vorher zur nächst größeren IEEE-Zahl gerundet werden. Für $f(x) = \operatorname{arsinh}(x)$ lässt sich dies durch die beiden folgenden Deklarationen

   ```
   interval X1(7.2075E-16,7.2075E-16);
   real Oberschranke(Sup(X1)); // Oberschranke >= 7.2075E-16
   ```

   sehr effektiv realisiert.

2. Zur Realisierung von Intervallfunktionen benötigt man nach (3.19),(3.20) von Seite 78 die Faktoren

   $$\frac{\texttt{Z1m}}{\texttt{N1m}} \leq 1 - \frac{\varepsilon(f)}{1+\varepsilon(f)} \quad \text{und} \quad 1 + \frac{\varepsilon(f)}{1-\varepsilon(f)} \leq \frac{\texttt{Z1p}}{\texttt{N1p}},$$

   wobei die Schranken `Z1m/N1m` und `Z1p/N1p` selbst wieder IEEE-Konstanten sind, deren Zähler und Nenner nach Seite 81 mit Hilfe der **C-XSC** Funktion `eps2fractions()` berechnet werden. Die IEEE-Zahlen `Z1m,N1m,Z1p,N1p` sind in Tabelle C.1 für jede Funktion $f(x)$ angegeben. Für $f(x) = \operatorname{arsinh}(x)$ gilt:

   $$\frac{\texttt{Z1m}}{\texttt{N1m}} = \frac{9007199254740982.0}{9007199254740992.0}, \qquad \frac{\texttt{Z1p}}{\texttt{N1p}} = \frac{4503599627370503.0}{4503599627370496.0}$$

3. Die relativen Fehlerschranken $\varepsilon(f)$ aus Tabelle C.1 werden nach dem Beispiel von Seite 81 benötigt, um Intervallfunktionen zu realisieren. Die in **C-XSC** vorhandenen Funktionen $\sqrt{1+x^2}$, $\sqrt{x^2+y^2}$ lassen sich durch naive Intervallrechnung jedoch direkt und effektiv darstellen, so dass deren Fehlerschranken nicht benötigt werden. Daher fehlen diese Funktionen in der obigen Tabelle.

Index

−>, 372

abs_relh(Modul), 30, 37, **38**, 40, 147, 152, 158
  abs_exp_abs(), 152
  abs_exp_rel(), 152
  abs_ln_abs(), 158
  abs_ln_rel(), 158
  abs_lnp1_abs(), 165
  abs_lnp1_rel(), 169
  abs_sqrt_abs(), 147
  abs_sqrt_rel(), 147
  eps_2(), 136
  eps_3(), 136
  eps_4(), 136
  eps_pow(), 136
  eps_rezi(), 136
  eps_ZdN(), 136
  rel_acosh_abs(), 178
  rel_acosh_rel(), 178
  rel_exp_abs(), 152
  rel_exp_rel(), 152
  rel_ln_abs(), 158
  rel_ln_rel(), 158
  rel_lnp1_abs(), 174
  rel_lnp1_rel(), 177
  rel_sqrt_abs(), 147
  rel_sqrt_rel(), 147
abs_relm(Modul), 30, 31, **33**
acoshp1, 331
addd(), 58, 163
addu(), 30, 58
Akkumulator, 261, 269
Algorithmus
  optimieren, 78, 121, 130, 204, 219, 285, 305, 319, 335
Approx-Error, 190, 194, 199, 381
Approximationsfehler, 179, 180, 196, 208, 210, 211
Arithmetik
  hochgenau, 15, **35**, 63
  maximalgenau, 15
Auslöschung, 185, 187, 213, 288, 289, 292, 308, 361, 367
Auswertefehler
  Polynome
    Binäre Koeffizienten, 102
    Rationale Koeffizienten, 93, 94, 98, 104
  Rationale Funktionen, 110
  Spezielle Terme, 121

Bernoulli-Zahlen, 359
bnd_arh(Modul), 66–68, 74, 81
bnd_util(Modul), 44, 72, 82, 85
  eps2fractions(), 78, 276
  frac_tools(), 130
  Horn_relh1(), 95
  Horn_relh1_Xi(), 95
  IEEE2frac_int(), 10
  Koeff_Rat(), 94
  Koeff_tools(), 95
  Max_bnd_Xi(), 72, 269
  Poly_Dat_rel(), 94
  Polynomgrad(), 94
  Term_relh(), 122

`comp`, 50
`coth`, 357
`Cut26`, 289

Denormalisierter Bereich, 3, **5**, 15, 375, 379
Dezimale Werte
    gerichtete Rundung, 103, 137, 330
`divd()`, 166
`divu()`, 31, 32
Doppelte Genauigkeit, 79, 121, 319, 330
double, *siehe* IEEE-double Format

$e(x)$, 48
e^x, 192
$e^x - 1$, 132, 134
e^{-x^2}, 316
Eingabe dezimaler Werte, 103, 137, 330
`eps2fractions`, 77, 78, 276, 403
$\mathrm{erf}(x)$, 194
`expmx2`, 316
`expo`, 6, 48
Exponent zur Basis 2, 48
Exponentialfunktion, 192
    $e^x - 1$, 132, 134

faithful, 207
Fehler
    absolute, 12
    Konvertierung, 6
    relative, 12
    verbesserte, 47
Fehlerhafte Funktionsargumente, 142
    $\ln(1 + x)$, 161
    $\ln(x)$, 155
    $\sqrt{x}$, 145
    e^x, 149
Fehlerkalkül: Produkt, Quotient, 135
Fehlerschranken
    absolute, 24
    Berechnung, 30, **40**, 203, 375, 378
    der Standardfunktionen, 401
    Grundoperationen, 14, 15
    relative, 26
Festkommadarstellung, 12
`Fixed`, 12
Fortgepflanzter Datenfehler, 24, 27
    absoluter, 24
    relativer, 27
`frac_tools`, 85, 130
Funktionen
    monotone, 75
Funktionsargumente, *siehe* Fehlerhafte Funktionsargumente

Gaußsche Fehlerfunktion, 194
Genauigkeit, 131, 353, 355
    doppelte, 79, 121, 319, 330
Geometrische Reihe, 183
Gerichtete Rundung, 19, 374
    $->$, 372
    `addd()`, 58, 163
    `addu()`, 30, 58
    `divd()`, 166
    `divu()`, 31, 32
    `muld()`, 177, 237
    `mulu()`, 95
    `subd()`, 163
    `subu()`, 321
Gesamtfehler, 26, 40, 179
    absoluter, 26
    relativer, 29
Gitterabstand, 4, 5
Gleitkommasysteme, 1
Globale Optimierung, 371

hidden bit, 5, 121
Hilfsfunktion, 195, 197

Horner-Schema, 90, 91, 201, 317, 335, 360

IEEE-double Format, **4**
IEEE-Zahlen speichern, **10**, 12
`IEEE2frac_int`, 10
Intervallauswertung, 183
Intervalleingabe, 23, 313
Intervallfunktion, 7, 77, 78, 214
  $\ln(1+x)$, 7
Intervalloperanden, 22
Intervallunterteilung, 43, 70, 82

`Koeff_tools`, 95
Konstruktor, 20
Konversionsfehler, 213
Konvertierungsfehler, 6
Konvexe Funktionen, 371

`l_interval`, 351
`l_real`, 347
Leibnizkriterium, 182, 183
Leibnizreihe, 332, 359
`ln_sqrtx2y2`, 219
`lnp1`, 7, 160

Manipulator, 12, 44
`mant`, 6
Mantisse, 2, 3, 5, 6
  abschließende Nullen, 48
  führende Nullen, 48
`Max_bnd_Xi`, 44, **72**, 82, 269
Maximumeinschließung, 377
`MaxReal`, 5
`Min-Incl`, 377, 393
Minimumeinschließung, 371, 377
`MinReal`, 5
`minreal`, 5
Module
  `abs_relh`, 30, **38**
  `abs_relm`, 30, **33**
  `bnd_arh`, 66–68, 74, 81
  `bnd_util`, 44, 72, 82, 85
Monotone Funktionen
  fallend, 75
  wachsend, 75
`muld()`, 177, 237
`mulu()`, 95

Nachfolger, 6
namespace
  cxsc, 11, 358
  std, 11
  taylor, 381
Newton-Schritt, 285, 311, 324
Normalisierter Bereich, **2**–**4**
Null-Intervall, 351

Oberschranke, 9

`Poly1_relh`, 94, 95
`Poly_relh`, 102
Polynome, *siehe* Auswertefehler
Präzision, 20, 131, 347, 351, 353, 355
`pred`, **7**, 50
Programme
  `Approx-Error`, 185, 190, 192, 381
  `Min-Incl`, 377, 393
  `Poly1_relh`, 93, 98
  `Poly_relh`, 102, 105, 108
  `Rat_F`, 114
  `Rat_relh`, 110, 113
Punktintervall, 20, 31, 50, 216, 277, 351, 366, 370

`Rat_F`, 114
`Rat_relh`, 110
Rationale Approximation, 185, 196
  Hilfsfunktion, 195, 197
Rationale Funktion, 109, 184, 192, 195
`Rnd_1pxh`, 66
`RndNext`, 19, 213, 326, 330

`RndUp`, 44
Rundung
  gerichtete, 19, 178
    −>, 372
    `addd()`, 58, 163
    `addu()`, 30, 58
    `divd()`, 166
    `divu()`, 31, 32
    `muld()`, 177, 237
    `mulu()`, 95
    `subd()`, 163
    `subu()`, 321
Rundungsfehler, 24, 25, 28
Rundungsfehlerfreie Operationen, 48
  Addition, 48
  Beispiele, 50–54
  Division, 49
  Multiplikation, 49
  Subtraktion, 49

$S(2, 53)$, 6, 13, 17
`Scientific`, 366
`SetDotPrecision`, 20, 348, 352, 366, 370
`SetPrecision`, 11, 12
Skalierung, 319, 355, 367
`sqr2uv`, 319
`sqrt1mx2`, 305, 367
`sqrtp1m1`, 212
`sqrtx2m1`, 285
Staggered Correction Arithmetik, 347
`stagprec`, 347, 349
`StagPrec()`, 347, 349
`subd()`, 163
`subu()`, 321
`succ`, **7**, 44, 46, 72, 73

Tastatureingabe, 330
Taylorarithmetik, 188, 191
Taylorentwicklung, 79
Taylorpolynom, 182
Taylorreihe, 89, 183, 195
`Term_relh`, 122
`times2pown`, 367

Unterlaufbereich, 13, 15, 379
Unterschranke, 8, 9

Vorgänger, 6

Wertebereich, 22, 29, **203**, 219
worst case error, 18

z_{lead}, **48**
z_{trail}, **48**

Literaturverzeichnis

[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions, Graphs, and Mathematical Tables*. Dover Publikations, INC., NEW YORK, 1972. ISBN 0-486-61272-4.

[2] G. Alefeld and J. Herzberger. *Einführung in die Intervallrechnung*. Reihe Informatik 12. **BI** Bibliographisches Institut Mannheim Wien Zürich, 1974. ISBN 3-411-01466-0.

[3] A. Bantle and W. Krämer. Ein Kalkül für verlässliche absolute und relative Fehlerabschätzungen. Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, 1998. Preprint Nr. 98/5.

[4] A. Bantle and W. Krämer. Automatic Forward Error Analysis for Floating-Point Algorithms. *Reliable Computing*, 7, No. 4:321–340, 2001.

[5] K. Braune. Standard Functions for Real and Complex Point and Interval Arguments with Dynamic Accuracy. *Computing Supplementum*, 6:159–184, 1988.

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

[7] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *C++ Toolbox for Verified Computing, Basic Numerical Problems*. Springer-Verlag Berlin Heidelberg, 1995.

[8] J. Herzberger. Topics in Validated Computations. Poceedings of the IMACS-GAMM International Workshop on Validated Numerics, Oldenburg, 1993. North Holland, 1994.

[9] N.J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.

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

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

[12] W. Hofschuster and W. Krämer. A Computer Orientated Approach to Get Sharp Reliable Error Bounds. *Reliable Computing*, 3:239–248, 1997.

[13] W. Hofschuster and W. Krämer. Eine schnelle und portable Funktionsbibliothek für reelle Argumente und reelle Intervalle im IEEE-Double-Format. Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, 1998. Preprint Nr. 98/7.

[14] W. Hofschuster, W. Krämer, S. Wedner, and A. Wiethoff. C-XSC 2.0, A C++ Class Library for Extended Scientific Computing. Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung, 2001. Preprint Nr. 2001/1.

[15] M. Iri. Simultaneous Computing of Functions, Partial Derivatives and Error Estimates of Rounding Errors. *Japan J. Appl. Math.*, 1:223–252, 1984.

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

[17] Konrad Königsberger. *Analysis*, volume 1. Springer-Verlag, Berlin Heidelberg New York, 1990. ISBN 3-540-52006-6.

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

[19] Walter Krämer. Inverse Standardfunktionen für reelle und komplexe Intervallargumente mit a priori Fehlerabschätzungen für beliebige Datenformate. Dissertation, Universität Karlsruhe, 1987.

[20] Walter Krämer. Mehrfachgenaue reelle und intervallmäßige Staggered-Correction Arithmetik mit zugehörigen Standardfunktionen. Report of the Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1988.

[21] Walter Krämer. Sichere und genaue abschätzung des Approximationsfehlers bei rationalen Approximationen. Report of the Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1996.

[22] Walter Krämer. A priori Worst Case Error Bounds for Floating-Point Computations. *Proceedings of the 13th IEEE Symp. on Computer Arithmetic, Asilomar, California*, pages 64–71, 1997.

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

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

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

[26] H.J. Stetter. Staggered Correction Representation, a Feasible Approach to Dynamic Precision. Proceedings of the Symposium on Scientific Software, edited by Cai, Fosdick, Huang, China University of Science and Technology Press, Beijing, China, 1989.

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

[28] A. Wiethoff. Ein Modul für hochgenaue Rechnerarithmetik über höheren Datentypen in C++. Diplomarbeit, Institut für Angewandte Mathematik, Universität Karlsruhe, Germany, 1990.