Zur Berechnung von Funktionswerteinschließungen bei speziellen Funktionen der mathematischen Physik

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der Fakultät für Mathematik der
Universität Karlsruhe (TH)

genehmigte

Dissertation

von

Dipl.-Math. oec. Werner Hofschuster

aus Heidenheim/Brenz

Tag der mündlichen Prüfung:	5. Juni 2000
Referent:	Prof. Dr. W. Krämer
Korreferent:	Prof. Dr. E. Kaucher

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik der Universität Karlsruhe (TH). An dieser Stelle möchte ich deshalb Herrn Prof. Dr. U. Kulisch danken, der mir die Möglichkeit gegeben hat, an seinem Institut zu arbeiten und diese Arbeit anzufertigen. Auch hat er die Entstehung der Arbeit mit großem Interesse begleitet.

Mein besonderer Dank gilt meinem Referenten Herrn Prof. Dr. W. Krämer, der auch die Thematik dieser Arbeit angeregt hat. Er hat durch seine ständige Gesprächs- und Diskussionsbereitschaft wesentlich zum Fortgang der Arbeit beigetragen.

Weiter danke ich Herrn Prof. Dr. E. Kaucher für die freundliche Übernahme des Korreferats.

Mein Dank gilt auch den Mitarbeiterinnen und Mitarbeitern des Instituts für Angewandte Mathematik, die durch ihre langjährige gute Zusammenarbeit und ihre Hilfsbereitschaft ihren Teil zur Entstehung dieser Arbeit beitrugen. Besonders erwähnt sei hier Herr Dr. D. Ratz, der mir dankenswerterweise geeignete „Stylefiles“ zum Textsatzsystem LaTeX überlassen hat.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Einleitung

Die mathematische Modellierung von Problemstellungen aus dem Bereich der Physik und der Ingenieurwissenschaften erfolgt häufig mittels Differentialgleichungen. Treten bei der Spezifikation oder bei der Lösung der Differentialgleichungen spezielle Funktionen, wie z.B. Gammafunktion, Fehlerfunktion oder Besselfunktionen auf, werden entsprechende Funktionen in der verwendeten Programmierumgebung auf dem Rechner benötigt. Um Lösungsverfahren aus dem Bereich der Verifikationsnumerik einsetzen zu können, müssen für diese auf dem Rechner bereitgestellten Funktionen mathematisch gesicherte (relative) Fehlerschranken bekannt sein.

Da die Herleitung der Fehlerschranken von Hand sehr mühsam und fehleranfällig, bei größeren Problemen in der Praxis sogar unmöglich ist, wird hierzu ein Kalkül für automatische Fehlerabschätzungen auf dem Rechner hergeleitet und verwendet.

Am Beispiel der Besselfunktionen werden bereits vorhandene Implementierungen untersucht und auf ihre Brauchbarkeit im Zusammenhang mit der Verifikationsnumerik überprüft. Die Entwicklung einer verläßlichen speziellen Funktion mit einer mathematisch gesicherten Fehlerschranke wird in Einzelschritten aufgezeigt.

Die vorliegende Arbeit ist wie folgt gegliedert: Im ersten Kapitel erfolgt eine kurze Einführung in die verwendeten Hilfsmittel aus dem Bereich der Verifikationsnumerik. Wichtige Eigenschaften der zugrundeliegenden Intervallrechnung werden in Anhang A zusammengefaßt.

Im zweiten Kapitel wird der Kalkül für automatische Fehlerabschätzungen vorgestellt. Nach dem Aufzeigen der Problematik werden die benötigten Formeln hergeleitet und bewiesen. Zudem wird die Umsetzung in eine geeignete Softwareumgebung beschrieben.

Die bei der numerischen Berechnung von Funktionen mittels geeigneter Approximationen entstehenden Approximationsfehler werden mit einer Methode erfaßt, die im dritten Kapitel vorgestellt wird.

Ein Überblick über verschiedene spezielle Funktionen und deren Implementierungen wird im vierten Kapitel gegeben. Es wird hier auch der Frage nachgegangen, inwieweit vorhandene Funktionsbibliotheken in der Numerik mit Ergebnisverifikation eingesetzt werden können.

Das fünfte Kapitel beschäftigt sich mit den sogenannten Besselfunktionen. Nach einer kurzen Beschreibung der wichtigsten Eigenschaften dieser Funktionen werden die Einzelschritte aufgezeigt, mit deren Hilfe eine verläßliche Funktion auf dem Rechner bereitgestellt werden kann. Am Beispiel des „Besselschen Irrgartens“ (Abschnitt 5.4.6.1, Seite 108) wird deutlich, daß eine naive Anwendung bekannter mathematischer Formeln auf dem Rechner nicht zum Ziel führt. Verschiedene Berechnungsmöglichkeiten werden hergeleitet und auf ihre Anwendbarkeit hin untersucht.

Im abschließenden sechsten Kapitel werden die Ergebnisse dieser Arbeit zusammengefaßt und ein Ausblick zur weiteren Entwicklung von speziellen Funktionen gegeben.

Karlsruhe, im Januar 2000

Kapitel 1

Hilfsmittel aus dem Bereich der Verifikationsnumerik

In diesem Kapitel werden mehrere Verfahren aus dem Bereich der Verifikationsnumerik, die in Kapitel 5 bei der Untersuchung von Methoden zur Berechnung von Funktionswerteinschließungen bei speziellen Funktionen der mathematischen Physik eingesetzt werden, bereitgestellt. Die verwendete Notation, sowie die im folgenden vorausgesetzten Grundlagen aus dem Bereich der Intervallrechnung sind in Anhang A zusammengefaßt.

1.1 Intervall–Langzahlarithmetiken

Beim Entwurf und bei der Implementierung von mathematischen Funktionen mit garantierter Ergebnisgenauigkeit werden Intervall–Langzahlarithmetiken benötigt. Mit ihrer Hilfe werden sowohl benötigte Konstanten mit beliebiger Genauigkeit berechnet, als auch Referenzfunktionen (siehe Abschnitt 5.2) implementiert, die zu Entwicklungs- und Testzwecken benötigt werden. Ein weiteres Einsatzgebiet ist die Approximationsfehlerbestimmung (vgl. Kapitel 3).

Im Rahmen dieser Arbeit werden dabei zwei verschiedene Arten von Intervall–Langzahlarithmetiken verwendet, die im folgenden kurz beschrieben werden sollen.

1.1.1 Intervall–Staggered Arithmetik

Es handelt sich hierbei um eine Intervall–Langzahlarithmetik, welche auf Gleitkommaoperationen aufbaut. Eine Beschreibung einer solchen Arithmetik findet sich z.B. in Krämer [59, 66], Lohner [78] oder in Krämer/Kulisch/Lohner [70]. Die wichtigsten Operationen sollen hier kurz vorgestellt werden.

S ist die Menge der Maschinenzahlen auf einem Rechner, mit $I\!I\!R$ werden die Intervalle über $I\!R$ und mit IS wird die Menge der Maschinenintervalle, d.h. Intervalle der Form $X = [\underline{x}, \overline{x}]$ with $\underline{x}, \overline{x} \in S$ bezeichnet.

Als Langzahlintervall („staggered Intervall“) x der Stufe n wird dann eine Summe von n Gleitkommazahlen $x_i \in S, i = 1(1)n,\ n \geq 0$, und einem zusätzlichen Gleitkommaintervall $X \in IS$ bezeichnet:

$$x = \sum_{i=1}^{n} x_i + X \tag{1.1}$$

Die Menge aller Langzahlintervalle der Stufe n wird mit IS_n bezeichnet.

In obiger Definition wird $n = 0$ ausdrücklich zugelassen. Die Langzahl-Intervallarithmetik der Stufe 0 entspricht gerade der normalen Intervallarithmetik, d.h. $IS_0 = IS$. In der verwendeten Pascal-XSC Implementierung wird aus Gründen einer einfacheren Realisierung allerdings $n \geq 1$ vorausgesetzt.

Das exakte Ergebnis einer arithmetischen Operation $\circ \in \{+, -, *, /\}$, (falls $0 \notin y$ für $\circ = /$) zweier Langzahlintervalle x und y wird wie in der Intervallarithmetik üblich definiert:

$$x \circ y := \{\xi \circ \eta \mid \xi \in x, \eta \in y\}.$$

Auf dem Rechner muß diese Ergebnismenge eingeschlossen werden. Dies kann mit Hilfe der genauen Skalarproduktoperation und eines langen Akkumulators, wie z.B. von Pascal–XSC oder C–XSC bereitgestellt, sehr einfach durchgeführt werden.

Da für die arithmetischen Operationen Langzahlintervalle unterschiedlicher Stufe als Operanden zugelassen werden sollen und das Ergebnis als Langzahlintervall mit einer vom Benutzer vorgegebenen Stufe berechnet werden soll, ist folgendes Vorgehen zweckmäßig: Es wird zunächst eine Unter- und eine Oberschranke des exakten Ergebnisses berechnet. Beide werden in einem langen Akkumulator abgespeichert. Das Ergebnis wird dann mittels einer Rundungsfunktion in einem Langzahlintervall gewünschter Stufe bereitgestellt. Für die Division muß von diesem Vorgehen abgewichen werden, hier wird ein iteratives Verfahren verwendet.

Die benötigte Rundungsfunktion Rundung kann in Form eines Algorithmus angegeben werden. Eingangsgrößen sind die beiden langen Akkumulatoren *down* und *up*, die die Unter- bzw. Oberschranke des exakten Ergebnisses enthalten, sowie die gewünschte Stufe n des Langzahlergebnisintervalls x. Die Ausgangsgröße des Algorithmus ist das Langzahlergebnisintervall x. Die im Algorithmus vorkommenden Größen xd, xu und x_i sind Gleitkommazahlen, X ist ein Intervall, i und n sind ganzzahlige Größen und *stop* ist eine logische Variable.

Der entsprechende Rundungsmodus für den Inhalt des langen Akkumulators wird durch die Symbole $\Box, \bigtriangledown, \bigtriangleup$, d.h. Rundung zur nächsten, zur nächstkleineren bzw. nächstgrößeren Gleitkommazahl, vorgegeben.

Algorithmus 1.1: Rundung $(down, up, n; x)$
Rundungsfunktion für Langzahlergebnisintervall

1. $stop := false$
2. $i := 0$
3. **repeat**
   $i := i + 1$
   $xd := \Box down$
   $xu := \Box up$
   if $xd = xu$ **or** $xd = pred(xu)$
   then $x_i := xd$
   $down := down - xd$
   $up := up - xd$
   else $x_i := 0$
   $stop := true$
   until $stop$ **or** $i = n$
4. **for** $i := i + 1$ **to** n **do** $x_i := 0$
5. $X := [\bigtriangledown lo, \bigtriangleup up]$

Bei der Addition bzw. Subtraktion von Langzahlintervallen werden die Unter- und Obergrenzen jeweils in eine langen Akkumulator addiert bzw. subtrahiert. Das Ergebnis wird dann mit Hilfe der Funktion Rundung in einem Langzahlintervall zur Verfügung gestellt.

Für die Addition ergibt sich damit

Algorithmus 1.2: Addition $(x, y; z)$
Addition zweier Langzahlintervalle

1. $up := \sum_{i=1}^{n_x} x_i + \sum_{i=1}^{n_y} y_i$
2. $down := up + \underline{X} + \underline{Y}$
3. $up := up + \overline{X} + \overline{Y}$
4. $z :=$ Rundung$(\ down,\ up\ )$

und analog für die Subtraktion

Algorithmus 1.3: Subtraktion $(x, y; z)$
Subtraktion zweier Langzahlintervalle

1. $up := \sum_{i=1}^{n_x} x_i \ - \ \sum_{i=1}^{n_y} y_i$
2. $down := up + \underline{X} - \overline{Y}$
3. $up := up + \overline{X} - \underline{Y}$
4. $z :=$ Rundung$(\ down,\ up\ )$

Die Multiplikation kann auf verschiedene Weisen implementiert werden. Aufgrund des Subdistributivitätsgesetzes der Intervallrechnung erhält man dabei im allgemeinen auch verschiedene Ergebnisse. Für die Langzahlintervalle $x = \sum_{i=1}^{n_x} x_i + X$ und $y = \sum_{j=1}^{n_y} y_j + Y$ gilt wegen der Subdistributivität

$$\begin{aligned} xy &= (\sum_{i=1}^{n_x} x_i + X) \cdot (\sum_{j=1}^{n_y} y_j + Y) \\ &\subseteq \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} x_i y_j + X \cdot \sum_{j=1}^{n_y} y_j + Y \cdot \sum_{i=1}^{n_x} x_i + XY \\ &\subseteq \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} x_i y_j + \sum_{j=1}^{n_y} X y_j + \sum_{i=1}^{n_x} Y x_i + XY. \end{aligned} \tag{1.2}$$

Im folgenden Multiplikationsalgorithmus wird die dritte Zeile von (1.2) zur Umsetzung verwendet.

Algorithmus 1.4: Multiplikation $(x, y; z)$
Multiplikation zweier Langzahlintervalle

1. $down := \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} x_i y_j$
2. $up := down$
3. $[down, up] := [down, up] + \sum_{j=1}^{n_y} X y_j + \sum_{i=1}^{n_x} Y x_i + XY$
4. $z :=$ Rundung$(\ down,\ up\ )$

Zur Division zweier Langzahlintervalle x und y wird ein iterativer Algorithmus eingesetzt, durch den nacheinander die n_z Komponenten z_i des Quotienten x/y bestimmt werden.

Zur Berechnung der Näherung $\sum_{i=1}^{n_z} z_i$ wird mit $z_1 = \Box m(x)/\Box m(y)$ gestartet; dabei bezeichnet $m(a)$ einen Punkt aus a, z.B. den Mittelpunkt und $\Box$ ist die Rundung zur nächstgelegenen Gleitkommazahl.

Durch Induktion läßt sich zeigen: Gegeben sei eine Näherung $\sum_{i=1}^{k} z_i$, dann kann der nächste Summand z_{k+1} mit

$$z_{k+1} = \Box \Big(\sum_{i=1}^{n_x} x_i - \sum_{i=1}^{n_y} \sum_{j=1}^{k} y_i z_j \Big) / \Box(m(y)), \tag{1.3}$$

berechnet werden. Bei der Ausführung auf dem Rechner ist dabei zu beachten, daß der Zähler mit Hilfe des exakten Skalarprodukts berechnet wird, d.h. das das exakte Ergebnis des Zählers mittels nur einer einzigen Rundung in das Gleitkommaraster S abgebildet wird. Der Nenner kann in normaler Gleitkommaarithmetik berechnet werden. Diese Art der Umsetzung auf dem Rechner garantiert, daß sich die z_i nicht überlappen, da der Defekt (d.h. der Zähler in (1.3)) einer jeden Näherung $\sum_{i=1}^{k} z_i$ mit einer einzigen Rundung berechnet wird.

Die Intervallkomponente Z des Langzahlintervalls z kann wie folgt berechnet werden:

$$Z = \Diamond \Big(\sum_{i=1}^{n_x} x_i - \sum_{i=1}^{n_y} \sum_{j=1}^{n_z} y_i z_j + X - \sum_{j=1}^{n_z} z_j Y \Big) / \Diamond(y), \tag{1.4}$$

das Symbol $\Diamond$ bezeichnet dabei die Intervallrundung in IS.

Nun sieht man, daß das Langzahlintervall $z = \sum_{i=1}^{n_z} z_i + Z$, das mit den Formeln (1.3) und (1.4) berechnet wurde, eine Obermenge des exakten Wertebereichs $\{\xi/\eta \mid \xi \in x, \eta \in y\}$ ist, denn für alle $\alpha \in X, \beta \in Y$ gilt:

$$\frac{\sum\limits_{i=1}^{n_x} x_i + \alpha}{\sum\limits_{i=1}^{n_y} y_i + \beta} = \sum_{j=1}^{n_z} z_i + \frac{\sum\limits_{i=1}^{n_x} x_i + \alpha - \sum\limits_{i=1}^{n_y}\sum\limits_{j=1}^{n_z} y_i z_j - \sum\limits_{j=1}^{n_z} z_j \beta}{\sum\limits_{i=1}^{n_y} y_i + \beta}.$$

Eine Intervallauswertung dieses Ausdrucks für $\alpha \in X$ und $\beta \in Y$ liefert sofort, daß die exakte Ergebnismenge von x/y im Langzahlintervall z enthalten ist.

Der Algorithmus zur Division zweier Langzahlvariablen x und y besteht damit aus drei Schritten:

Algorithmus 1.5: Division $(x, y; z)$ Division zweier Langzahlintervalle
1. $z_1 := m(x)/m(y)$ 2. Berechnung der Komponenten z_{k+1} mit (1.3), $k = 0(1)n_z - 1$ 3. Berechnung der Intervallkomponente Z mit (1.4)

Das Ergebnis steht dann als Langzahlintervall $z = \sum_{i=1}^{n_z} z_i + Z$ zur Verfügung.

1.1.2 Intervall–Langzahlarithmetik auf Integer-Basis

Die im Abschnitt 1.1.1 beschriebene Intervall–Langzahlarithmetik beruht vollständig auf einer Gleitkommaarithmetik. Dem Vorteil einer einfachen Realisierung und eines effizienten Laufzeitverhaltens steht der große Nachteil eines eingeschränkten Exponentenbereichs gegenüber. In diesem Abschnitt wird deshalb eine andere Realisierungsmöglichkeit einer Intervall–Langzahlarithmetik vorgestellt.

Die abzuspeichernde Langzahl wird hierbei in Vorzeichen, Mantisse und Exponent zerlegt. Die Mantisse wird dann als Feld von ganzen Zahlen (integer–Größen) dargestellt, der Exponent und das Vorzeichen werden getrennt mitgeführt.

Es sind heute eine Reihe von Langzahlpaketen verfügbar (einige davon sind im Internet frei erhältlich). Die meisten dieser Pakete enthalten allerdings keine Langzahlintervallarithmetik. Deshalb wird im folgenden das `mp`–Paket[1] für Pascal–XSC verwendet. Dieses Paket besteht aus zwei Pascal–XSC Modulen: Modul `mp_ari` für Langzahlarithmetik und Modul `mpi_ari` für Langzahlintervallarithmetik. Diese beiden Module dienen im wesentlichen als Schnittstelle zu den eigentlichen Langzahlroutinen, die in der Programmiersprache C realisiert sind (Krämer [63, 65]) und sich im Laufzeitsystem von Pascal–XSC (Cordes [32]) befinden. Aufgrund der Vielzahl der in beiden Modulen vordefinierten Operatoren und Funktionen, sowie des Operatorkonzepts von Pascal–XSC führt der Einsatz dieses Langzahlpakets zur recht einfachen und übersichtlichen Programmen.

Der Datentyp für Langzahlen heißt `mpreal`, der Datentyp für Langzahlintervalle `mpinterval`. Dieser setzt sich wie bei Intervallen üblich zusammen:

```
global mpinterval = global record inf, sup: mpreal end;
```

Eine Variable von Datentyp `mpreal` ist aus Pascal–XSC Sicht ein Pointer auf einen `integer`-Datentyp, auf C Ebene handelt es sich um einen Verbunddatentyp („struct“) mit mehreren Komponenten (Exponent, dynamisch lange Mantisse, Längenfeld, verschiedene Flags). Die Mantisse wird bezüglich der Basis $B = 2^{32}$ dargestellt, d.h. jede Mantissenziffer belegt 32 Bit Speicherplatz. Eine positive `mpreal`–Zahl $x \neq 0$ kann in der Form

$$x = \sum_{n=0}^{l_{\max}-1} m_n B^{ex-n}\,, \quad m_0 \neq 0,\ m_n \in \{0,1,\ldots,2^{32}-1\} \tag{1.5}$$

geschrieben werden. Der Exponent ex variiert dabei im Bereich von $e_{\min} := -2^{31}$ bis $e_{\max} := 2^{31} - 1$.

Alle vordefinierten arithmetischen Operationen verarbeiten Operanden mit unterschiedlicher Anzahl an Mantissenziffern. Die gewünschte Anzahl an Mantissenziffern für das Ergebnis kann für jede Operation getrennt gesetzt werden. Dies geschieht mit der Funktion `setprec(n)`, der Wert `n` gibt dabei die Anzahl der Mantissenziffern an.

In der aktuellen Compilerversion von Pascal–XSC muß die Speicherverwaltung durch den Benutzer erfolgen, d.h. Langzahlen und Langzahlintervalle müssen von Hand initialisiert und freigegeben werden (Hammer [41]). Dies geschieht mit Hilfe der folgenden Prozeduraufrufe:

[1] `mp` steht für multiple precision

`mpinit`: Allokieren von Speicherplatz für Variablen vom Typ `mpreal` oder `mpinterval` in Abhängigkeit von der aktuell gesetzten Anzahl von Mantissenziffern,

`mpfree`: Freigeben von Langzahlvariablen,

`mpvlcp`: Erzeugen einer temporären Kopie eines Funktions-, Prozedur- oder Operatorparameters,

`mptemp`: Markieren einer Langzahlvariable als temporäre Variable.

Die Ergebnisvariable einer Funktions- oder Operatordefinition kann aufgrund einer Beschränkung durch den Pascal-Standard nicht mit Hilfe der oben genannten Prozeduren erfolgen. Deshalb geschieht die Zuweisung an den Variablennamen einer Funktion mit Hilfe eines kleinen programmiertechnischen Tricks. Es werden hierfür Zuweisungsoperatoren mit den folgenden rechten Seiten zur Verfügung gestellt:

InitFunc, InitOp: zur Allokierung der Ergebnisvariable

TempFunc, TempOp: zur Markierung der Ergebnisvariable als temporäre Variable.

Bei der Erstellung eines Programms mit Hilfe der Langzahlarithmetikmodule `mp_ari` and `mpi_ari` müssen deshalb die folgenden fünf Regeln R1 – R5 beachtet werden.

R1: **Variablen** vom Typ `mpreal` oder `mpinterval` müssen durch Aufruf von `mpinit` initialisiert und sollten durch Aufruf von `mpfree` freigegeben werden:

```
VAR
  x: mpreal;
  y: mpinterval;
BEGIN
  mpinit(x); mpinit(y);
  ...
  mpfree(x); mpfree(y);
END;
```

R2: **Wertparamter** vom Typ `mpreal` oder `mpinterval` müssen durch Aufruf von `mpvlcp` als Kopie gesichert und anschließend durch Aufruf von `mpfree` rekonstruiert werden:

```
PROCEDURE DoSomething(x: mpreal);
BEGIN
  mpvlcp(x);
  ...
  mpfree(x);
END;
```

R3: **Referenzparamter** dürfen weder mit `mpinit` initialisiert noch mit `mpfree` freigegeben werden.

R4: Die **Ergebnisvariable einer Funktion** vom Typ `mpreal` oder `mpinterval` muß durch Zuweisung des Wertes `InitFunc` initialisiert werden und sollte durch Zuweisung des Wertes `TempFunc` freigegeben werden:

```
FUNCTION ComputeSomething(...): mpreal;
BEGIN
  ComputeSomething:= InitFunc;
  ...
  ComputeSomething:= TempFunc;
END;
```

R5: Die **Ergebnisvariable eines Operators** vom Typ `mpreal` oder `mpinterval` muß durch Zuweisung des Wertes `InitOp` initialisiert werden und sollte durch Zuweisung des Wertes `TempOp` freigegeben werden:

```
OPERATOR + (...) add: mpreal;
BEGIN
  add:= InitOp;
  ...
  add:= TempOp;
END;
```

Weitere Informationen zu dieser Langzahlarithmetik können Krämer [64, 66], Hofschuster [48] sowie Krämer/Kulisch/Lohner [70] entnommen werden.

1.2 Differentiationsarithmetik, Taylorarithmetik

Im Bereich des wissenschaftlichen Rechnens sind zur Berechnung von Funktionsableitungen drei Verfahren gebräuchlich: die numerische Differentiation, die symbolische Differentiation und die *automatische Differentiation*. Während bei der numerischen Differentiation die Werte der Ableitungen durch Differenzenquotienten approximiert werden, ermittelt die symbolische Differentiation zunächst durch Anwendung der Differentiationsregeln explizite, im allgemeinen recht umfangreiche Darstellungen für die Ableitungen, die dann für verschiedene Argumente ausgewertet werden können. Unter Verwendung einer sogenannten Differentiationsarithmetik kann auf dem Rechner die automatische Differentiation angewendet werden, bei der parallel zur Anwendung der Differentiationsregeln auch das Einsetzen der Argumente erfolgt, so daß lediglich Zahlenwerte und keine symbolischen Formeln bearbeitet werden müssen. Die Berechnung der Ableitungen erfolgt automatisch mit der Berechnung eines Funktionswertes.

Innerhalb der automatischen Differentiation unterscheidet man die sogenannte *Vorwärtsmethode*, die ausführlich z. B. in Rall [101] behandelt wird, und die sogenannte *Rückwärtsmethode*, auch schnelle automatische Differentiation genannt. Letztere wird auf das Lösen spezieller Gleichungssysteme zurückgeführt und bringt einerseits einen deutlichen Laufzeitgewinn mit sich, erfordert andererseits jedoch einen erhöhten Speicherbedarf und eine aufwendigere Programmierung. Einen tieferen Einblick in die Rückwärtsmethode bietet z. B. Fischer [38]. Wir wollen im folgenden kurz die prinzipielle Methode der automatischen Differentiation anhand der in der Vorwärtsmethode verwendeten Differentiationsarithmetik erläutern. Wir behandeln dabei die Gradientenarithmetik und die Hessematrixarithmetik für Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ mit dem Ziel, jeweils Intervalleinschließungen für die berechneten Werte zu erhalten. Somit werden alle Operationen unter Verwendung von Intervallarithmetik ausgeführt.

Die Gradientenarithmetik ist eine Arithmetik für Paare der Form

$$U = (u, u^g), \quad \text{mit } u \in I\mathbb{R} \text{ und } u^g \in I\mathbb{R}^n,$$

die jeweils mit u den Funktionswert der Funktion $U(x)$ und mit $u^g = \nabla U(x)$ den Wert des Gradienten an der festen Stelle $x \in I\mathbb{R}^n$ repräsentieren. Für die konstante Funktion $U(x) = c$ ist somit $U = (u, u^g) = (c, 0)$, und für die für die Funktion $U(x) = x_i$ definiert man $U = (u, u^g) = (x_i, e^i)$, wobei e^i den i-ten Einheitsvektor bezeichnet.

Die Definition der arithmetischen Operationen für zwei Paare $U = (u, u^g)$ und $V = (v, v^g)$ erfolgt durch

$$\begin{aligned} U &+ V &&= (u + v,\ u^g + v^g), \\ U &- V &&= (u - v,\ u^g - v^g), \\ U &\cdot V &&= (u \cdot v,\ v \cdot u^g + u \cdot v^g), \\ U &/ V &&= (u/v,\ (v \cdot u^g - u \cdot v^g)/v^2), \quad v \neq 0. \end{aligned}$$

Für eine Elementarfunktion $S \in \mathcal{SF}$ mit $S : I\!R \to I\!R$ und $U = (u, u^g)$ definiert man

$$S(U) = (S(u),\ S'(u) \cdot u^g).$$

Dabei ist S' die explizit bekannte Ableitung der Funktion S.

Die Hessematrixarithmetik ist eine Arithmetik für Tripel der Form

$$U = (u, u^g, u^h), \quad \text{mit } u \in I\!R,\ u^g \in I\!R^n \text{ und } u^h \in I\!R^{n \times n},$$

die jeweils mit u den Funktionswert der Funktion $U(x)$, mit $u^g = \nabla U(x)$ den Wert des Gradienten und mit $u^h = \nabla^2 U(x)$ den Wert der Hessematrix an der festen Stelle $x \in I\!R^n$ repräsentieren. Für die konstante Funktion $U(x) = c$ ist somit $U = (u, u^g, u^h) = (c, 0, 0)$, und für die Funktion $U(x) = x_i$ definiert man $U = (u, u^g, u^h) = (x_i, e^i, 0)$.

Die arithmetischen Operationen für zwei Tripel $U = (u, u^g, u^h)$ und $V = (v, v^g, v^h)$ sind in den Komponenten w und w^g von $W = U \circ V$ für $\circ \in \{+, -, \cdot, /\}$ definiert wie in der Gradientenarithmetik, während sich die dritten Komponenten gemäß

$$\begin{aligned} W &= U + V \implies w^h = u^h + v^h, \\ W &= U - V \implies w^h = u^h - v^h, \\ W &= U \cdot V \implies w^h = u \cdot v^h + u^g \cdot (v^g)^T + v^g \cdot (u^g)^T v \cdot u^h, \\ W &= U / V \implies w^h = (u^h - w^g \cdot (v^g)^T - v^g \cdot (w^g)^T - w \cdot v^h)/v, \quad v \neq 0 \end{aligned}$$

berechnen. Für eine Elementarfunktion $S \in \mathcal{SF}$ mit $S : I\!R \to I\!R$ und $U = (u, u^g, u^h)$ definiert man

$$S(U) = (S(u),\ S'(u) \cdot u^g,\ S'(u) \cdot u^h + S''(u) \cdot u^g \cdot (u^g)^T).$$

Dabei sind S' und S'' die explizit bekannten ersten und zweiten Ableitungen der Funktion S.

1.3 Erweiterte Intervallarithmetik

Bei der Nullstelleneinschließung mittels eines Intervall-Newton-Schrittes (Abschnitt 1.4) werden Intervallausdrücke der Form

$$r - A/B$$

mit $r \in \mathbb{R}$ und $A, B \in I\mathbb{R}$ verwendet, in denen ein Intervall B auftritt, das möglicherweise die 0 enthält. Die Behandlung dieses Falles erfordert den Einsatz einer erweiterten Intervallarithmetik. In der Literatur finden sich verschiedene Ansätze für eine solche Arithmetik, wir wollen jedoch hier lediglich auf die eingeschränkte Erweiterung der in unserem Fall benötigten Operationen − und / eingehen (vgl. auch Hansen [42], Kaucher [54] oder Moore [94]). Im folgenden werden die Bezeichnungen

$$\begin{aligned} I\mathbb{R}_\infty &:= I\mathbb{R} \cup \{(-\infty, \rho] \mid \rho \in \mathbb{R}\} \cup \{[\lambda, \infty) \mid \lambda \in \mathbb{R}\} \cup \{(-\infty, \infty)\}, \\ IR_\infty &:= IR \cup \{(-\infty, \rho] \mid \rho \in R\} \cup \{[\lambda, \infty) \mid \lambda \in R\} \cup \{(-\infty, \infty)\} \end{aligned}$$

verwendet.

Definition 1.3.1 *Ein Paar $W = (W^1 \,|\, W^2) := W^1 \cup W^2$ von Intervallen $W^1, W^2 \in I\mathbb{R}_\infty \cup \{\emptyset\}$ heißt erweitertes Intervallpaar, wenn entweder gilt $\sup W^1 < \inf W^2$ oder $W^2 = \emptyset$.*

Der Durchmesser eines erweiterten Intervallpaares $W = (W^1 \mid W^2)$ mit endlichen Komponenten $W^1, W^2 \in I\mathbb{R} \cup \{\emptyset\}$ ist definiert durch

$$d(W) = d(W^1) + d(W^2), \quad \text{mit } d(\emptyset) = 0.$$

Seien $A, B \in I\mathbb{R}$, dann definieren wir den erweiterten Intervallquotienten $W = A/B$ für $0 \in B$ durch (vgl. auch Ratz [104])

$$W := \begin{cases} (-\infty, \infty) & \text{für } \underline{B} = \overline{B} = 0 \\ (-\infty, \infty) & \text{für } \underline{A} = \overline{A} = 0 \\ (-\infty, \infty) & \text{für } \underline{A} < 0 < \overline{A} \\ [\overline{A}/\underline{B}, \infty) & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ \underline{B} < \overline{B} = 0 \\ (-\infty, \overline{A}/\overline{B}] \cup [\overline{A}/\underline{B}, \infty) & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ \underline{B} < 0 < \overline{B} \\ (-\infty, \overline{A}/\overline{B}] & \text{für } \underline{A} < \overline{A} \le 0 \ \wedge \ 0 = \underline{B} < \overline{B} \\ (-\infty, \underline{A}/\underline{B}] & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ \underline{B} < \overline{B} = 0 \\ (-\infty, \underline{A}/\underline{B}] \cup [\underline{A}/\overline{B}, \infty) & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ \underline{B} < 0 < \overline{B} \\ [\underline{A}/\overline{B}, \infty) & \text{für } 0 \le \underline{A} < \overline{A} \ \wedge \ 0 = \underline{B} < \overline{B} \end{cases}$$

und für $0 \notin B$ durch (A.1).

Darüber hinaus definieren wir die Differenz $U = r - W$ einer reellen Zahl r und eines erweiterten Intervallpaares W durch

$$U = \begin{cases} (-\infty, \infty) & \text{für } W = (-\infty, \infty) \\ (-\infty, r - \lambda] & \text{für } W = [\lambda, \infty) \\ [r - \rho, \infty) & \text{für } W = (-\infty, \rho] \\ (-\infty, r - \lambda] \cup [r - \rho, \infty) & \text{für } W = (-\infty, \rho] \cup [\lambda, \infty) \\ [r - \rho, r - \lambda] & \text{für } W = [\lambda, \rho] \end{cases}$$

Von den Verbandsoperationen benötigen wir für unsere Anwendungen außerdem die Schnittbildung eines erweiterten Intervallpaares mit einem Standard-Intervall, die wir für $W^1, W^2 \in I\!I\!R_\infty$ und $X \in I\!I\!R$ durch

$$W \cap X := (W^1 \cap X \mid W^2 \cap X)$$

definieren. Beim Übergang auf eine Rechenanlage verlangen wir auch von den erweiterten Intervalloperationen die Definition über den Semimorphismus, so daß für die Maschinenoperation $\diamondsuit$ und $\diamondsuit$ gilt

$$\begin{aligned} A \diamondsuit B &= \Diamond(A/B), \\ r \diamondsuit U &= \Diamond(r - U), \end{aligned}$$

wobei $A, B \in IR$, $r \in R$ und U ein erweitertes Intervallpaar mit $U^1, U^2 \in IR_\infty$ ist. Die Rundung für erweiterte Intervallpaare definieren wir durch

$$\Diamond(W) = \begin{cases} (-\infty, \infty) & \text{für } W = (-\infty, \infty) \\ [\bigtriangledown(\lambda), \infty] & \text{für } W = [\lambda, \infty) \\ (-\infty, \bigtriangleup(\rho)] & \text{für } W = (-\infty, \rho] \\ (-\infty, \bigtriangleup(\rho)] \cup [\bigtriangledown(\lambda), \infty) & \text{für } W = (-\infty, \rho] \cup [\lambda, \infty) \\ [\bigtriangledown(\lambda), \bigtriangleup(\rho)] & \text{für } W = [\lambda, \rho] \end{cases}$$

für $W^1, W^2 \in I\!I\!R_\infty$, also $\lambda, \rho \in I\!R$.

1.4 Langzahlmäßiges Intervall–Newton–Verfahren

Zur Einschließung von Funktionsnullstellen wird in dieser Arbeit das erweiterte Intervall-Newton-Verfahren im $I\!R^1$ angewandt (vgl. Kapitel 5.5). Zum

besseren Verständnis wird dieses Verfahren kurz angegeben und damit auch die unmittelbare Anwendung der erweiterten Intervallarithmetik demonstriert (vgl. auch Ratz [103]). Das herkömmliche Intervall-Newton-Verfahren (vgl. z. B. Alefeld/Herzberger [2], [3] oder Mayer [90]) zur Bestimmung der Nullstelle x^* der stetig differenzierbaren Funktion $f : \mathbb{R} \to \mathbb{R}$ ist ausgehend von einer Starteinschließung X^0 mit $0 \notin F'(X^0)$ definiert durch

$$X^{k+1} := \left(m(X^k) - \frac{f(m(X^k))}{F'(X^k)} \right) \cap X^k, \quad \text{für } k = 0, 1, 2, \ldots$$

Das Verfahren kann wegen der verwendeten Schnittbildung nicht divergieren, es ist jedoch aufgrund der Voraussetzung $0 \notin F'(X^0)$ nicht anwendbar, wenn im Startintervall X^0 mehrere Nullstellen von f liegen. Unter Verwendung der erweiterten Intervalloperationen aus dem letzten Abschnitt läßt sich dieser Mangel beseitigen, und es ist möglich, *alle* Nullstellen im Startintervall zu bestimmen. Die dabei durch die Division möglicherweise entstehenden erweiterten Intervallpaare werden durch die Schnittbildung auf ein Paar von Intervallen aus $I\mathbb{R}$ reduziert, die im Anschluß jeweils getrennt weiter iteriert werden können.

Der rekursiv formulierte Algorithmus 1.6 beschreibt das erweiterte Intervall-Newton-Verfahren im $\mathbb{R}^1$ unter Verwendung eines erweiterten Intervallpaares $W = (W^1 \,|\, W^2)$. Ist X die Starteinschließung für die Nullstellen der Funktion f, und gilt für die Lösungsmenge L zunächst $L = \{\,\}$, so liefert $\mathsf{Newton}\,(X, eps, L)$ als Ergebnis die Menge

$$L = \{X \in I\mathbb{R} \mid 0 \in F(X) \ \wedge \ d(X) < eps\}.$$

Algorithmus 1.6: Newton (X, eps, L)
Erweitertes Intervall-Newton-Verfahren

1. **if** $0 \notin F(X)$ **then return**.
2. $c := m(X)$; $z := f(c)$; $G := F'(X)$.
3. $W := (c - z/G) \cap X$.
4. **if** $W^1 = W^2 = \emptyset$ **then return**.
5. **if** $W^1 = X$ **then**
   $W^1 := [\underline{X}, c]$; $W^2 := [c, \overline{X}]$.
6. **if** $W^1 \neq \emptyset$ **then**
   if $d(W^1) < eps$ **then** $L := L \cup \{W^1\}$
   else Newton (W^1, eps, L).
7. **if** $W^2 \neq \emptyset$ **then**
   if $d(W^2) < eps$ **then** $L := L \cup \{W^2\}$
   else Newton (W^2, eps, L).

Ähnlich wie das klassische Newton-Verfahren kann auch das Intervall-Newton-Verfahren geometrisch interpretiert werden. In jedem Iterationsschritt werden an der Stelle $m(X)$ zwei Geraden an den Graphen der Funktion f angelegt werden. Es handelt sich dabei um die Geraden mit der Steigung $\underline{F'(X)}$ bzw. $\overline{F'(X)}$, also mit der kleinsten bzw. größten Steigung von f im Intervall X. Ihre Schnittpunkte mit der x-Achse entprechen den Intervallgrenzen der neuen Iterierten vor der Intervallschnittbildung mit der alten Iterierten X. In einem Newton-Schritt wird also zunächst das Intervall

$$Y := m - f(m)/F'(X) = [\lambda, \rho]$$

berechnet. Die im Anschluß daran erfolgende Schnittbildung liefert somit

$$X := Y \cap X = [\lambda, \rho] \cap [l, r] = [l, \rho]$$

als neue Iterierte. Im Fall $0 \in F'(X)$ schneidet die x-Achse in ρ, die Gerade mit der größten Steigung schneidet sie in λ.

In einem (erweiterten) Newton-Schritt wird also zunächst das erweiterte Intervallpaar

$$W := m - f(m)/F'(X) = (-\infty, \rho] \cup [\lambda, \infty)$$

berechnet und die anschließende Schnittbildung liefert

$$X := W \cap X = ((-\infty, \rho] \cup [\lambda, \infty)) \cap [l, r] = [l, \rho] \cup [\lambda, r],$$

also ein erweitertes Intervallpaar als neue Iterierte, dessen Teilintervalle jeweils getrennt weiter zu iterieren wären.

1.5 Gleichungssystemlöser für schwach besetzte Matrizen

Gelöst werden soll das lineare (Intervall-)Gleichungssystem $Ax = b$ mit einer Bandmatrix $A \in I\!I\!R^{n\times n}$ und $b, x \in I\!I\!R^n$. Die Matrix A habe l Nebendiagonalen unterhalb und k Nebendiagonalen oberhalb der Hauptdiagonalen ($0 \leq l, k < n$ und $l + k > 0$). Das Gleichungssystem hat damit folgende Gestalt:

$$\begin{pmatrix} a_{1,1} & \cdots & a_{1,k+1} & & 0 \\ \vdots & \ddots & & \ddots & \\ a_{l+1,1} & & \ddots & & a_{n-k,n} \\ & \ddots & & \ddots & \vdots \\ 0 & & a_{n,n-l} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix} \tag{1.6}$$

mit $a_{ij}, b_i, x_i \in I\!I\!R$, $i, j = 1(1)n$, d.h. als Koeffizienten können sowohl reelle Zahlen, als auch reelle Intervalle vorkommen.

Eine bekannte Möglichkeit (vgl. Krämer/Kulisch/Lohner [70], Lohner [79]) Einschließungen für die Lösung eines Gleichungssystem zu berechnen, besteht darin, zunächst eine Näherungslösung $\tilde{x}$ und eine Näherungsinverse R zu bestimmen. Ausgehend vom Fehler $y = x - \tilde{x}$ von der wahren Lösung x gilt die Gleichung

$$Ay = b - A\tilde{x},$$

die mit der Näherungsinversen R multipliziert in der Form

$$y = R(b - A\tilde{x}) + (I - RA)y$$

geschrieben werden kann. Mit $f(y) := R(b - A\tilde{x}) + (I - RA)y$ ergibt sich die Fixpunktgleichung

$$y = f(y) \tag{1.7}$$

für den Fehler y. Aus (1.7) kann die folgende Iterationsformel abgeleitet werden, die mittels Intervallarithmetik berechnet wird:

$$y_{k+1} = R \diamondsuit(b - A\tilde{x}) + \diamondsuit(I - RA)y_k \tag{1.8}$$

Das Symbol $\diamondsuit$ bezeichnet hierbei Operationen, die aufgrund möglicher Auslöschung von führenden Ziffern auf dem Rechner mit Hilfe des exakten Skalarprodukts berechnet werden müssen.

Beim Vorliegen einer Matix A mit Bandstruktur sollte die direkte Berechnung einer Näherungsinverse R vermieden werden. Stattdessen wird für die Näherungsinverse der Ansatz

$$R := (LU)^{-1} \approx A^{-1} \tag{1.9}$$

verwendet, hierbei ist

$$LU \approx A \tag{1.10}$$

eine näherungsweise LU-Zerlegung der Matrix A. Die Matrizen L und U sind hierbei wiederum Bandmatrizen. Die aus der Fixpunktgleichung (1.7) abgeleitete Iterationsformel hat nun die Form

$$y_{k+1} = (LU)^{-1}(b - A\tilde{x}) + (I - (LU)^{-1}A)y_k. \tag{1.11}$$

Nach Multiplikation mit LU erhält man

$$LUy_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)y_k. \tag{1.12}$$

Somit sind zwei lineare Gleichungssysteme mit jeweils einer Bandmatrix U bzw. L in Dreiecksgestalt zu lösen. In jedem Iterationsschritt wird zunächst eine Lösungseinschließung z_{k+1} von

$$Lz_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)y_k$$

berechnet und anschließend eine Einschließung für y_{k+1} aus

$$Uy_{k+1} = z_{k+1}$$

bestimmt. Wichtig ist, daß bei der intervallmäßigen Berechnung aufgrund des sogenannten „wrapping–Effekts“ nicht einfach eine Vorwärts- bzw. Rückwärtssubstitution erfolgt.

Die hierzu verwendete Idee wird anhand des folgenden Gleichungssystem mit einer unteren (Band-)Dreiecksmatrix A aufgezeigt:

$$\begin{pmatrix} a_{1,1} & & & & 0 \\ \vdots & \ddots & & & \\ a_{m,1} & & \ddots & & \\ & \ddots & & \ddots & \\ 0 & & a_{n,n-m+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix} \tag{1.13}$$

mit $a_{ij}, b_i, x_i \in I\!I\!R$, $i, j = 1(1)n$.

Das Gleichungssystem (1.13) ist äquivalent mit der linearen Differenzengleichung der Ordnung m

$$a_{i,i-m+1}x_{i-m+1} + \cdots + a_{i,i}x_i = b_i \ , \quad i = m, \ldots, n, \tag{1.14}$$

mit den Startwerten

$$x_i = (b_i - a_{i,i-1}x_{i-1} - \cdots - a_{i,1}x_1)/a_{i,i} \ , \quad i = 1, \ldots, m-1 \ . \tag{1.15}$$

Mit den Vektoren

$$y_i := (x_i, \ldots, x_{i+m-1})^T \in I\!R^m, \quad i \geq 0, \tag{1.16}$$

$$d_i := (0, \ldots, 0, b_i/a_{i,m})^T \in I\!R^m, \quad i \geq 0, \tag{1.17}$$

und der $m \times m$-Matrix

$$A_i := \begin{pmatrix} 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & 0 & & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ \dfrac{-a_{i,0}}{a_{i,m}} & \dfrac{-a_{i,1}}{a_{i,m}} & \cdots & \dfrac{-a_{i,m-2}}{a_{i,m}} & \dfrac{-a_{i,m-1}}{a_{i,m}} \end{pmatrix} \in I\!R^{m \times m} \ . \tag{1.18}$$

kann diese Differenzengleichung in der Form

$$y_{i+1} = A_i y_i + d_i, \quad i \geq 0 \tag{1.19}$$

mit den Startwerten

$$y_0 = (x_0, \ldots, x_{m-1})^T \ , \tag{1.20}$$

geschrieben werden.

Die Gleichung (1.19) kann im Raum $I\!R^m$ geometrisch als affine Abbildung von y_i nach y_{i+1} interpretiert werden. Bei der Berechnung in Intervallarithmetik tritt hierbei der „wrapping–Effekt“ auf. Ein Intervallvektor $[y_i]$ wird durch (1.19) auf die Menge

$$\{A_i y_i + d_i | y_i \in [y_i]\},$$

d.h. auf ein Parallelepiped abgebildet.

Auf dem Rechner kann ein Parallelepiped am einfachsten durch eine (reguläre) Basismatrix B, einen Intervallvektor $[z]$ und einen Punktvektor $\tilde{y}$ dargestellt werden. Die Punktmenge

$$P := P(B, [z], \tilde{y}) := \{\tilde{y} + Bz | z \in [z]\} \tag{1.21}$$

ist dann ein Parallelepiped mit einer Ecke $\tilde{y}$. Eine etwas einfachere Form erhält man, wenn man den Punktvektor $\tilde{y}$ in einem entsprechend modifizierten Intervallvektor $[z]$ mitberücksichtigt:

$$P := P(B, [z]) := \{Bz | z \in [z]\} \tag{1.22}$$

Ob die Darstellung (1.21) oder (1.22) besser geeignet ist, hängt von der Größenordnung der tatsächlichen Werte ab. Im folgenden wird aufgrund der einfacheren Berechenbarkeit die Darstellung (1.22) verwendet. Mit Hilfe der Parallelepipede sollen bei diesem Verfahren kleine Fehlerterme eingeschlossen werden, deshalb wird eine Lösung y_i von (1.19) als Summe einer Gleitkommaapproximation $\tilde{y}_i$ und eines Fehlers $\hat{y}_i$, der durch ein Intervall $[\hat{y}_i]$ eingeschlossen wird geschrieben:

$$y_i = \tilde{y}_i + \hat{y}_i \ . \tag{1.23}$$

Mit der Approximation

$$\tilde{y}_{i+1} \approx A_i \tilde{y}_i + d_i \tag{1.24}$$

ergibt sich dann

$$\hat{y}_{i+1} = A_i \hat{y}_i + d_i + (A_i \tilde{y}_i - \tilde{y}_{i+1}) \ . \tag{1.25}$$

Als Startwert ($i = 0$) wird, falls y_0 ein Vektor mit Gleitkommazahlelementen ist, $\tilde{y}_0 = y_0$ gewählt; andernfalls wird y_0 entsprechend gerundet und der Rundungsfehler $\hat{y}_0$ wird durch einen Intervallvektor $[\hat{y}_0] = \diamondsuit(y_0 - \tilde{y}_0)$ eingeschlossen.

Die Berechnung einer Einschließung für den Fehler $\hat{y}_{i+1}$ wird nun so durchgeführt, daß das Ergebnis jedes Iterationsschrittes in einem Parallelepiped eingeschlossen werden kann, d.h. $\hat{y}_{i+1} \in P(B_{i+1}, [z_{i+1}])$. Hierzu wird in jedem Schritt eine neue Basismatrix B_{i+1} zur Darstellung des entsprechenden Parallelepipeds eingeführt. Mit

$$\hat{d}_i = d_i + (A_i \tilde{y}_i - \tilde{y}_{i+1})$$

kann die Iteration (1.25) durch

$$\begin{aligned} [z_{i+1}] &:= (B_{i+1}^{-1} A_i B_i)[z_i] + B_{i+1}^{-1} \hat{d}_i, \quad i \geq 0 \\ [\hat{y}_{i+1}] &:= B_{i+1}[z_{i+1}] \end{aligned} \tag{1.26}$$

ersetzt werden. B_{i+1} ist hierbei eine reguläre Matrix und $B_0 := I$, wenn $\tilde{y}_0 \neq y_0$ wird $[z_0] := [\hat{y}_0]$ gesetzt, sonst $[z_0] = 0$.

In jedem Iterationsschritt wird nun allerdings die Inverse der neuen Basismatrix B_{i+1} benötigt. Da bei Rechung mit Gleitkommaarithmetik diese Inversen im allgemeinen nicht exakt dargestellt werden können, muß stattdessen eine Einschließung von B_{i+1}^{-1} berechnet werden.

Wir fordern nun, daß die Basismatrix B_{i+1} orthogonal und somit regulär ist. Erreicht wird dies, indem zuerst eine QR–Zerlegung der Mittelpunktsmatrix $m(\diamondsuit A_i B_i)$ durchgeführt wird und dann der orthogonale Anteil Q als neue Basismatrix verwendet wird:

$$B_{i+1} := Q_{i+1}, \quad \text{mit} \quad \tilde{B}_{i+1} = m(\diamondsuit A_i B_i), \quad \tilde{B}_{i+1} \approx Q_{i+1} R_{i+1} \ . \tag{1.27}$$

Für die praktische Durchführung genügt es, wenn die QR–Zerlegung näherungsweise in Gleitkommaarithmetik berechnet wird, die neue Basismatrix B_{i+1} ist dann eine fast orthogonale Matrix. Diese Eigenschaft von B_{i+1} hat Vorteile bei der Berechnung einer Einschließung der inversen Matrix von B_{i+1}. Für eine orthogonale Matrix gilt $Q^{-1} = Q^T$, d.h. für eine fast orthogonale Matrix ist $Q^T \approx Q^{-1}$ eine gute Approximation der Inversen.

Für eine fast orthogonale Matrix Q gilt

$$||I - QQ^T||_\infty = q << 1 \ .$$

Eine Einschließung der Inversen $[Q^{-1}]$ von Q erhält man nun leicht aus der Neumannschen Reihe:

$$Q^{-1} \in [Q^{-1}] := Q^T + [-\epsilon, \epsilon] \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \tag{1.28}$$

mit

$$\epsilon = \frac{q}{1-q} \ ||Q^T||_\infty \ . \tag{1.29}$$

Die berechneten Intervalle hängen von der Reihenfolge der Zeilen der Matrix A_i ab. Dies kann bei der Implementierung durch die Einführung einer zusätzlichen Permutationsmatrix verhindert werden. Das prinzipielle Vorgehen hierzu kann dem Algorithmus 1.7 (Spaltentausch) entnommen werden.

Algorithmus 1.7: Spaltentausch (B, z)

Sortieren der Spalten von B nach abnehmender Länge der Seiten von $P(B, z)$

1. {Berechnung der Seitenlängen im Quadrat }
   for $i := 1$ **to** m **do** $length_i := (B_{*,i} \cdot B_{*,i}) \cdot \mathtt{diam}(z_i)^2$
2. {Sortieren der Spalten von B nach Seitenlängen }
   for $i := 1$ **to** $m - 1$ **do**
   for $j := i + 1$ **to** m **do**
   if $length_i < length_j$ **then**
   Vertausche $(B_{*,i}, B_{*,j})$
   Vertausche $(length_i, length_j)$
3. **return** B

Die näherungsweise QR–Zerlegung mit Householder–Matrizen wird nach dem folgenden Algorithmus in Gleitkommaarithmetik ausgeführt:

Algorithmus 1.8: QR (A)

Näherungsweise QR–Zerlegung von A

1. $Q := I$
2. **for** $k := 1$ **to** $m - 1$ **do**
   $b := A_{k..n,k}$ { b ist $(n - k + 1)$–Vektor }
   if $b \neq 0$ **then**
   if $A_{k,k} < 0$ **then** $s := \|b\|_2$ **else** $s := -\|b\|_2$
   $b_k := A_{k,k} - s$
   $s := 1/(s \cdot b_k)$
   {lösche Einträge unterhalb der Diagonale von A:}
   for $i := k + 1$ **to** n **do**
   $r := s \cdot (b \cdot A_{k..n,i})$
   for $j := k$ **to** n **do** $A_{j,i} := A_{j,i} + b_j \cdot r$
   for $i := 1$ **to** n **do**
   $r := s \cdot (Q_{i,k..n} \cdot b)$
   for $j := k$ **to** n **do** $Q_{i,j} := Q_{i,j} + b_j \cdot r$
3. QR $:= Q$

Die Berechnung einer Einschließung der Inverse einer fast orthogonalen Matrix Q wird durch den folgenden Algorithmus angegeben. Falls für die Norm

$q = ||I - Q^T Q||_\infty \geq 1$ gilt, wird das Fehlerflag err_code auf den Wert 1 gesetzt.

Algorithmus 1.9: $\mathsf{Inv}\,(Q, err\_code)$
Einschließung der Inverse einer fast orthogonalen Matrix Q

1. $q := \|I - Q^T Q\|_\infty$
   if $q \geq 1$ **then** $err\_code := 1$
   else $err\_code := 0$
   if $err\_code = 1$ **then return**
2. $\epsilon := \|Q^T\|_\infty \cdot q/(1-q)$
   for $i := 1$ **to** m **do**
   for $j := 1$ **to** m **do**
   $Q1_{i,j} := Q_{j,i} + [-\epsilon, \epsilon]\|Q^T\|_\infty$
3. $\mathsf{Inv} := Q1$

Die Berechnung eines Schrittes der Iteration $y_{i+1} = Ay_i + d_i$ wird durch Algorithmus 1.10 wiedergegeben:

Algorithmus 1.10: $\mathsf{Iteration}\,(A, d, y0, B0, z0, y1, B1, z1)$
Ausführung eines Iterationsschrittes

1. {Berechnung der Approximation}
   $y1_{1..m-1} := y0_{2..m}$
   $y1_m := (m(b) - \sum_{i=1}^{m} m(a_{i-1}) \cdot y0_i)/m(a_m)$
2. {Einschließung des Fehlers}
   $B1 := m(A) \cdot B0$
   $\mathsf{Spaltentausch}\,(B1, z0)$
   $\mathsf{QR}\,(B1)$
   $BI1 := \mathsf{Inv}\,(B1)$
   $d := \diamondsuit(b - \sum_{i=1}^{m} a_{i-1} \cdot y0_i)/a_m - y1_m$
   $z1 := (BI1 \cdot (\mathsf{Coeff}\,(A) \cdot B0)) \cdot z0 + BI1_{*,m} \cdot d$
3. **return** $y1, B1, z1$

Der Nachweis der Existenz einer Lösung von (1.6) erfolgt mit Hilfe des Brouwerschen Fixpunktsatzes, im Programm wird hierzu die Inklusionsbedingung

$$y_{k+1} = F(y_k) \subset \overset{\circ}{y_k} \tag{1.30}$$

überprüft (vgl. Krämer/Kulisch/Lohner [70]).

Die Implementierung dieses Gleichungssystemlösers für Bandsysteme erfolgte in der Programmiersprache Pascal–XSC als Modul `bandlss`. Im Unterschied zu dem in Krämer/Kulisch/Lohner [70] angegebenen Programm darf die Matrix A des Gleichungssystems (1.6) Intervalleinträge enthalten. Im Rahmen dieser Arbeit wird dieser Gleichungssystemlöser zur Berechnung einer Lösungseinschließung für das in Abschnitt 5.4.6.3 aufgestellte Gleichungssystem verwendet.

Im Modul `bandlss` werden im wesentlichen die folgenden Routinen bereitgestellt:

Routine	Kurzbeschreibung
`min`, `max`	Hilfsroutinen zur Miniumum- bzw. Maximumbildung
`norm`	Spaltensummennorm einer Matrix
`inv_orth`	Inverse einer (fast) orthogonalen Matrix
`QR`	QR–Zerlegung mit Householder–Matrizen
`lss_triangular`	Vorwärtseinsetzen für unteres Dreieckssystem bzw. Rückwärtseinsetzen für oberes Dreieckssystem
`lss_lower`	Vorwärtseinsetzen mit Koordinatentransformation
`lss_upper`	Rückwärtseinsetzen mit Koordinatentransformation
`lu_decomp`	Näherungsweise LU–Zerlegung
`lss`	Löser für Bandsysteme
`read_Ab`	Hilfsroutine zum Abspeichern eines Bandsystems

Die in der Tabelle angegebenen Routinen stehen zum Teil im Modul `bandlss` mehrfach zur Verfügung, d.h. für Eingangs- bzw. Ausgangsgrößen unterschiedlicher Datentypen. Dank des Konzepts der Überladung von Funktions- und Prozedurnamen in Pascal–XSC können diese Routinen dennoch jeweils unter dem selben Namen aufgerufen werden.

Die quadratische $n \times n$–Bandmatrix A des linearen Gleichungssystems (1.6) wird, um insbesondere bei größeren Systemen Speicherplatz zu sparen,

in Form einer $n \times (l+k+1)$–Matrix $S = (s_{ij})$, $1 \leq i \leq n$, $-l \leq j \leq k$ mit

$$\begin{pmatrix} s_{1,-l} & \cdots & s_{1,k} \\ s_{2,-l} & \cdots & s_{2,k} \\ \vdots & & \vdots \\ s_{n-1,-l} & \cdots & s_{n-1,k} \\ s_{n,-l} & \cdots & s_{n,k} \end{pmatrix} := \begin{pmatrix} 0 & & a_{1,1} & a_{1,2} & \cdots & a_{1,k+1} \\ & \cdot & a_{2,2} & a_{2,3} & \cdot & \\ a_{l+1,1} & & \vdots & \cdot & & a_{n-k,n} \\ \vdots & & a_{n-1,n-1} & a_{n-1,n} & & \\ a_{n,n-l} & \cdots & a_{n,n} & & & 0 \end{pmatrix}$$

abgespeichert. Die Haupt- und Nebendiagonalen der Matrix A bilden dabei jeweils eine Spalte der Matrix S, die 0-te Spalte der Matrix S ergibt sich dabei aus der Hauptdiagonalen der Matrix A.

Der vollständige Quellcode der Implementierung ist im Anhang B.3 angegeben.

1.6 Verifizierte Integration

Für viele spezielle Funktionen der mathematischen Physik existiert eine Integraldarstellung. Beim Einsatz eines Verfahrens zur verifizierten Integration kann über eine dieser Integraldarstellungen eine Einschließung des Funktionswertes der zugehörigen speziellen Funktion berechnet werden (vgl. Abschnitt 5.4.2).

Eine einfache Möglichkeit zur Einschließung des Wertes eines bestimmten Integrals wird hier beschrieben (siehe auch Krämer/Kulisch/Lohner [70]). Gebräuchliche numerische Integrationsformeln zur Berechnung des Integralwertes

$$I = \int_a^b f(x)dx \tag{1.31}$$

haben die Form

$$I = J + R. \tag{1.32}$$

Das Integral I wird durch den ersten Ausdruck J approximiert. J ist dabei ein Skalarprodukt gewisser Gewichte w_i und Funktionswerte f an bestimmten Stellen x_i:

$$J = \sum_{i=0}^{n} w_i f(x_i) \ . \tag{1.33}$$

Der zweite Ausdruck R in (1.32) ist das Restglied der Integrationsformel. Für hinreichend glatte Integranden f kann das Restglied oft durch höhere

Ableitungen der Funktion f an gewissen (in der Regel unbekannten) Zwischenstellen des Integrationsbereichs $[a, b]$ ausgedrückt werden.

Integrationsformeln dieses Typs sind z.B. Newton-Côtes Formeln, die Romberg Integration oder die Gauß–Quadratur.

Das Grundprinzip für die verifizierte Berechnung von I ist bei allen diesen Formeln gleich: Es werden Einschließungen der Funktionswerte $f(x_i)$ berechnet und anschließend wird das Skalarprodukt J mittels Intervallarithmetik berechnet. Für das Restglied R werden mittels automatischer Differentiation (siehe Abschnitt 1.2) die höheren Ableitungen, die im Ausdruck von R vorkommen, berechnet. Der unbekannte Zwischenwert wird durch das Intervall $[a, b]$ ersetzt.

Der Einfachheit halber wird im folgenden nur die Simpsonregel als Beispiel für eine Newton-Côtes Formel betrachtet. Für eine ausreichend glatte Funktion f kann die Simpsonregel

$$I = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b)) \; - \; \frac{(b-a)^5}{90} f^{(4)}(\xi), \; \xi \in [a, b] \quad (1.34)$$

in einer iterativen Form benutzt werden. Hierzu wird das Integrationsintervall $[a, b]$ in n äquidistante Teilintervalle der Länge $h = (b-a)/n$ mit den Stützstellen $x_i = a+ih, i = 0(1)n$ unterteilt, hierbei wird n gerade vorausgesetzt. Die Integrationsregel wird dann auf jedes Teilintervall (bzw. Paar von Teilintervallen) getrennt angewandt. Die berechneten Approximationswerte und die Restglieder werden aufsummiert.

Bei Verwendung der Simpsonregel ergibt sich mit $n/2$ Teilintervallen der Länge $2h$

$$I = J_S + R_S \quad (1.35)$$

mit der Approximation

$$J_S = \frac{h}{3} \sum_{i=0}^{n} s_i f(x_i) \quad (1.36)$$

und dem Restglied

$$R_S = -\frac{h^5}{90} \sum_{i=1}^{n/2} f^{(4)}(\xi_i), \text{ mit } \xi_i \in [x_{2i-2}, x_{2i}]. \quad (1.37)$$

Die Berechnung erfolgt nun mittel Intervallarithmetik, für die jeweilige Approximation müssen Einschließungen $\diamondsuit f(x_i)$ der Funktionswerte von f an

den Stützstellen berechnet werden, anschließend wird J_S intervallmäßig ausgewertet:

$$J_S \in [J_S] := h \diamondsuit 3 \diamondsuit \diamondsuit \sum_{i=0}^{n} s_i \diamondsuit \diamondsuit f(x_i) \ . \tag{1.38}$$

Bessere Einschliesungen können erreicht werden, wenn die Summe der Formel (1.38) mit Hilfe eines langen Akkumulators exakt berechnet werden und anschließend nur einmal in das verwendete Gleitkommaraster gerundet werden:

$$J_S \in [J_S] := h \diamondsuit 3 \diamondsuit \diamondsuit \sum_{i=0}^{n} s_i \diamondsuit f(x_i) \ . \tag{1.39}$$

Einschließungen des jeweiligen Restgliedes erhält man, in dem man die Zwischenwerte ξ_i durch ein einschließendes Intervall, d.h. durch $[x_{2i-2}, x_{2i}]$ ersetzt:

$$R_S \in [R_S] := -h^5 \diamondsuit 90 \diamondsuit \diamondsuit \sum_{i=1}^{n/2} \diamondsuit f^{(4)}([x_{2i-2}, x_{2i}]) \tag{1.40}$$

Eine Einschließung für den Wert des Integrals I ergibt sich somit aus der Summe $[J_S] + [R_S]$.

Kapitel 2

Kalkül für automatische Fehlerabschätzungen

In diesem Abschnitt werden Techniken hergeleitet, welche es erlauben, Rundungsfehler in Gleitkommaalgorithmen sicher abzuschätzen. Auch die Fehlerfortpflanzung wird durch den hier vorgestellten Kalkül mit erfaßt. Die mit diesem Kalkül berechneten Schranken sind gleichmäßige a priori Fehlerschranken.

2.1 Fehlerarten

Im Bereich des „Wissenschaftlichen Rechnens“ werden konkrete Anwendungsprobleme mit Hilfe von modernen Rechnenanlagen gelöst. Hierbei läßt sich die Vorgehensweise in der Regel in mehrere Schritte gliedern: Zunächst wird ein mathematisches Modell formuliert, daß das Anwendungsproblem möglichst gut abbildet. Dieses mathematische Modell wird dann mit numerischen Methoden gelöst. Hierzu wird ein Algorithmus, oft unter Verwendung von Modellvereinfachungen, aufgestellt. Der Algorithmus wird dann in ein Progamm umgesetzt und es wird damit auf der Maschine ein numerisches Ergebnis des mathematischen Problems berechnet. Anschließend müssen diese numerischen Ergebnisse noch geeignet interpretiert werden, um eine Lösung des Anwendungsproblems zu erhalten.

In jedem dieser Schritte können Fehler entstehen: Modellfehler, Vereinfachungen aller Art, Meß- und Datenfehler, Approximationsfehler, Programmierfehler, Konversionsfehler, Rundungsfehler, Fehler bei der Interpretation der Ergebnisse.

Im Rahmen dieser Arbeit werden die mathematisch faßbaren Fehler näher untersucht. Dies sind zum einen die Approximationsfehler[1], die durch die verwendeten numerischen Verfahren bedingt sind. Zum anderen die Rundungsfehler, die bei der Berechnung mit endlicher Genauigkeit auf der Maschine unvermeidbar entstehen. Fehler, die bei der Rundung von Eingangsgrößen in das verwendete Gleitkommasystem entstehen, sollen ebenfalls mit erfaßt werden.

Grundsätzlich wird zwischen dem absoluten und dem relativen Fehler unterschieden:

Definition 2.1.1 *Sei $\tilde{x}$ eine Näherung für $x \in \mathbb{R}$. Dann heißt*

$$\Delta_x := \tilde{x} - x$$

absoluter Fehler und für $x \neq 0$ heißt

$$\varepsilon_x := \frac{\tilde{x} - x}{x}$$

relativer Fehler.

2.2 Grundidee des Fehlerkalküls

Bereits im Jahr 1966 stellte Moore [93] für die damaligen Rechenanlagen fest, daß bei vielen Millionen Rechenoperationen in einer Sekunde und komplizierteren Algorithmen mit Rechenzeiten von einigen Stunden eine Fehleranalyse von Hand völlig ausgeschlossen ist. Dies führte zu der Idee, die Fehleranalyse vom Rechner selbst durchführen zu lassen. Im Rahmen der Intervallarithmetik können Fehlerschranken zur Laufzeit eines Algorithmus mitberechnet werden. Das nachfolgend vorgestellte Fehlerkalkül zielt darauf ab, a priori Fehlerschranken auf dem Rechner zu ermitteln.

Grundlagen zur Abschätzung von Rundungsfehlern bei der Rechnung mit Gleitkommazahlen finden sich bei Wilkinson [123]. Ein systematisch aufgebautes Fehlerkalkül zur Bestimmung von a priori Fehlerschranken findet man bei Braune [24] und Krämer [58]. In Krämer [58, Anhang A] ist bereits ein Hinweis enthalten, daß langwierige Fehlerabschätzungen per Hand mittels eines Computerprogramms umgangen werden können. Der Einsatz der naiven Intervallrechnung ist hierzu nicht geeignet.

Im Zusammenhang mit der Entwicklung und Implementierungen von elementaren Funktionen (d.h. mathematische Standardfunktionen) wurde

[1] auch Verfahrensfehler

mit der Herleitung eines Kalküls für rechnergestützte Fehlerabschätzungen begonnen, näheres hierzu kann Hofschuster/Krämer [49, 50, 52], Bantle/Krämer [12] und Krämer [69] entnommen werden.

Die Grundidee des Fehlerkalküls soll an einem einfachen Beispiel aufgezeigt werden:

$$f(x) := \mathrm{ln1p}\left(x + \frac{x}{\sqrt{1 + \left(\frac{1}{x}\right)^2} + \frac{1}{x}}\right)$$

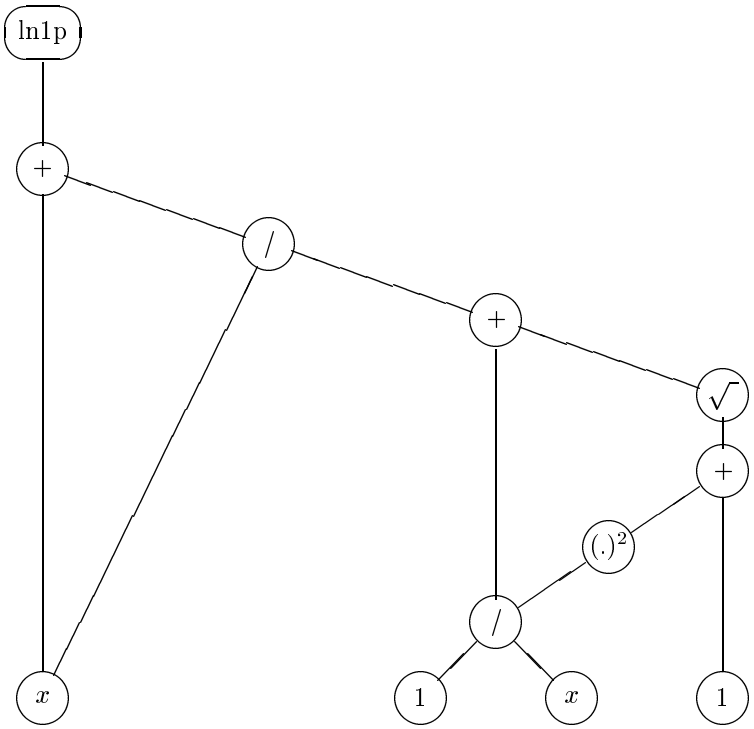


Abbildung 2.1: Ausdrucksbaum

Die reellwertige Funktion f wird an der Stelle x mittels des auf der rechten Seite stehenden Ausdrucks berechnet, dabei ist $\ln1p(x) := \ln(1+x)$, $x > -1$ eine auf dem Rechner vorhandene elementare Funktion. Die Berechnungsvorschrift wird durch den Ausdruckbaum in Abbildung 2.1 wiedergegeben.

Die Berechnung dieses Ausdrucks auf der Maschine erfolgt, indem die Werte x und c, die sich in den Blättern des Baumes befinden, in das Gleitkommaraster gerundet werden und auf diese gerundeten Werte $\tilde{x}$ und $\tilde{c}$ werden dann die entsprechend vorgegebenen Gleitkommaverknüpfungen angewandt. In diesem Beispiel bezeichnet die Konstante c den als Gleitkommazahl exakt darstellbaren Wert 1, im allgemeinen gilt $c \neq \tilde{c}$, für den zu berechnenden Funktionswert gilt $f(x) = f(x, c)$.

Zur Berechnung der gesuchten Fehlerschranke für $\tilde{f}(\tilde{x}, \tilde{c})$ wird ein neuer Verbunddatentyp eingeführt, dessen erste Komponente aus einem Intervall und dessen zweite Komponente aus einer reellen nichtnegativen Zahl besteht. Die erste Komponente enthält immer eine Einschließung des exakten Wertes, die zweite gibt eine Fehlerschranke für die zugehörige Gleitkommazahl wieder. Dies wird im nachfolgenden Baumdiagramm (Abbildung 2.2) verdeutlicht.

Alle Größen in den Blättern des Baumes (untere Zeile des Diagramms) sind bekannt ($[x] = [a, b]$, absolute Fehlerschranke $\Delta(x)$ mit $|x - \tilde{x}| \leq \Delta(x)$, Fehlerschranken für die Parameter). Die Berechnung erfolgt schrittweise von den Blättern zur Wurzel des Baumes. Als Ergebnis steht eine Intervalleinschließung I_{12} der exakten Ergebnismenge $\{f(x) | x \in [x]\}$ und eine Fehlerschranke Δ_{12} mit $|f(x) - \tilde{f}(\tilde{x})| \leq \Delta_{12}$ für alle $x \in [a, b]$ zur Verfügung.

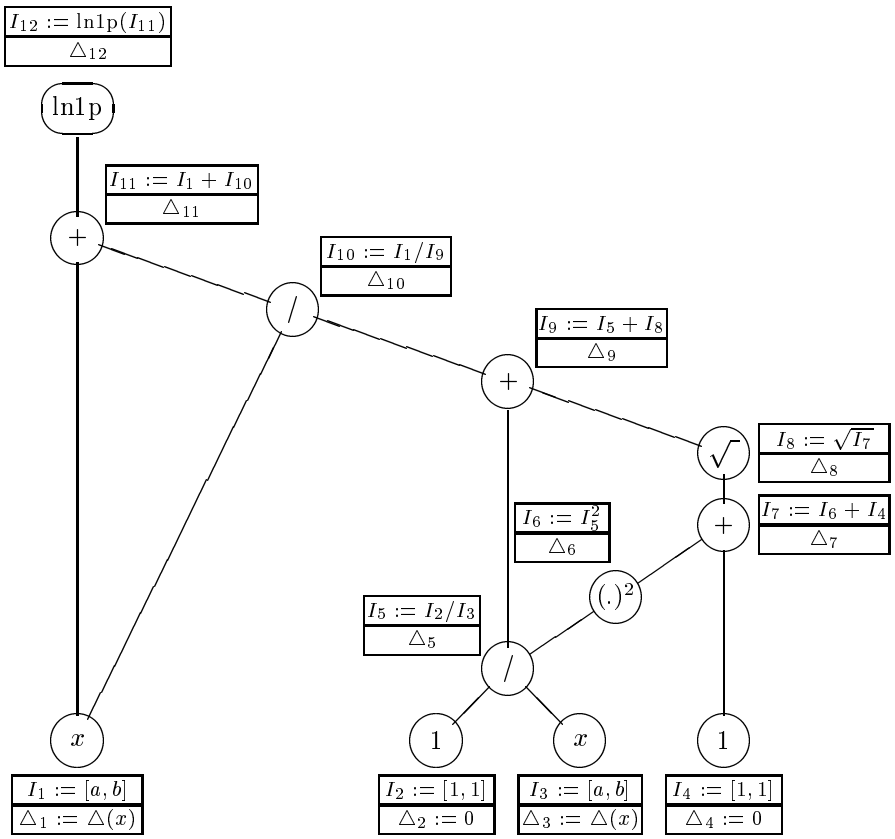


Abbildung 2.2: Ausdrucksbaum mit Fehlerschrankenberechnung

2.3 Fehlerschranken für die Grundoperationen

Für den Fehlerkalkül wird vorausgesetzt, daß die Operationen auf der Maschine dem IEEE-Standard [10, 19] entsprechen. Das bedeutet insbesondere, daß für die Grundoperation $\circ \in \{+, -, \cdot, /\}$ und deren Maschinenäquivalent

$\boxdot, \circ \in \{+, -, \cdot, /\}$ die folgende Beziehung gilt:

$$\bigwedge_{\circ \in \{+,-,\cdot,/\}} \quad \bigwedge_{\substack{a,b \in S \text{ mit} \\ |a \circ b| \in [\texttt{MinReal},\texttt{MaxReal}]}} \left| \frac{a \circ b - a \boxdot b}{a \circ b} \right| \leq \varepsilon^* . \quad (2.1)$$

Lemma 2.3.1 *(Unterlaufbereich)* Im Unterlaufbereich
$U := (-\texttt{MinReal},\texttt{MinReal})$ gilt die folgende Abschätzung:

$$\bigwedge_{\circ \in \{+,-,\cdot,/\}} \quad \bigwedge_{\substack{a,b \in S \\ |a \circ b| < \texttt{MinReal}}} |a \boxdot b - a \circ b| \leq d([0, \texttt{MinReal}]) \quad (2.2)$$

$$= \texttt{MinReal}. \quad (2.3)$$

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

Die obigen Beziehungen sind richtig, falls als Rundungsmode „Rundung zur nächsten Maschinenzahl“ verwendet wird. Bei Verwendung von gerichtet gerundeten Operationen muß $\varepsilon^* = \frac{1}{2}b^{1-l}$ in Formel (2.1) durch $2\varepsilon^* = b^{1-l}$ ersetzt werden. Dies trifft auch dann zu, wenn man von der Maschinenarithmetik nur voraussetzt, daß sie „faithful“ ist (siehe Definition A.2.5).

Die folgenden Schreibenweisen und Bezeichnungen werden bei der Herleitung des Kalküls verwendet:

$\circ \in \{+, -, \cdot, /\}, \quad a \in A \in I\!I\!R, \quad b \in B \in I\!I\!R$ sowie
$\tilde{a} = a + \Delta_a = a(1 + \varepsilon_a)$ mit $|\Delta_a| \leq \Delta(a), |\varepsilon_a| \leq \varepsilon(a) = |a|\Delta(a)$ und
$\tilde{b} = b + \Delta_b = b(1 + \varepsilon_b)$ mit $|\Delta_b| \leq \Delta(b), |\varepsilon_b| \leq \varepsilon(b) = |b|\Delta(b)$, also
$\tilde{a} \in \tilde{A} := A + [-\Delta(a), \Delta(a)] = A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]$,
$\tilde{b} \in \tilde{B} := B + [-\Delta(b), \Delta(b)] = B \cdot [1 - \varepsilon(b), 1 + \varepsilon(b)]$.

Dabei seien die Intervalle $A = [\underline{a}, \overline{a}]$ und $B = [\underline{b}, \overline{b}]$ $(\underline{a}, \overline{a}, \underline{b}, \overline{b} \in I\!R)$, die absoluten Fehlerschranken $\Delta(a)$ und $\Delta(b)$ und die relativen Fehlerschranken $\varepsilon(a)$ und $\varepsilon(b)$ gegeben.

Bemerkung: Falls eine Arithmetik mit „gradual underflow“ verwendet wird und die Arithmetik auch im Unterlaufbereich maximal genaue Grundoperationen zur Verfügung stellt, gilt offensichtlich

$$|\tilde{a} \circ \tilde{b} - \tilde{a} \boxdot \tilde{b}| \leq \texttt{dMinReal} \tag{2.4}$$

für alle $\circ \in \{+, -, \cdot, /\}$ und $\tilde{a}, \tilde{b} \in S$ mit $|\tilde{a} \circ \tilde{b}| < \texttt{MinReal}$.

2.3.1 Absolute Fehlerschranken

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

Für das Ergebnis der Maschinenverknüpfung $\tilde{a} \boxdot \tilde{b}$ soll nun eine Schranke $\Delta(\circ) \in I\!R^+$ berechnet werden, so daß

$$|(a \circ b) - (\tilde{a} \boxdot \tilde{b})| \leq \Delta(\circ)$$

für alle $a \in A$ mit $|a - \tilde{a}| \leq \Delta(a)$ bzw. $|a - \tilde{a}| \leq \varepsilon(a)|a|$ und für alle $b \in B$ mit $|b - \tilde{b}| \leq \Delta(b)$ bzw. $|b - \tilde{b}| \leq \varepsilon(b)|b|$ gilt.

2.3.1.1 Abschätzung des fortgepflanzten Datenfehlers

Satz 2.3.2 Für die Fortpflanzung des absoluten Datenfehlers $|(a \circ b) - (\tilde{a} \circ \tilde{b})|$ gilt bei exakter Rechnung

$$|(a+b) - (\tilde{a}+\tilde{b})| \leq \Delta(a) + \Delta(b) =: \Delta_{dat,+} \tag{2.5}$$

$$|(a-b) - (\tilde{a}-\tilde{b})| \leq \Delta(a) + \Delta(b) =: \Delta_{dat,-} \tag{2.6}$$

$$|(a \cdot b) - (\tilde{a} \cdot \tilde{b})| \leq \Delta(a)\Delta(b) + |a|\Delta(b) + |b|\Delta(a) =: \Delta_{dat,\cdot} \tag{2.7}$$

$$|(a/b) - (\tilde{a}/\tilde{b})| \leq \frac{\Delta(a) + |\frac{a}{b}|\Delta(b)}{|b| - \Delta(b)} =: \Delta_{dat,/} \text{ für } \Delta(b) < |b|\,. \tag{2.8}$$

Beweis:

$$
\begin{array}{lrcl}
Add.: & |(a+b)-(\tilde{a}+\tilde{b})| & = & |a+b-(a+\Delta_a+b+\Delta_b)| \\
 & & = & |\Delta_a+\Delta_b| \\
 & & \leq & \Delta(a)+\Delta(b)=\Delta_{dat,+}
\end{array}
$$

$$
\begin{array}{lrcl}
Sub.: & |(a-b)-(\tilde{a}-\tilde{b})| & = & |a-b-(a+\Delta_a-b-\Delta_b)| \\
 & & = & |\Delta_a-\Delta_b| \\
 & & \leq & \Delta(a)+\Delta(b)=\Delta_{dat,-}
\end{array}
$$

$$
\begin{array}{lrcl}
Mul.: & |(a\cdot b)-(\tilde{a}\cdot\tilde{b})| & = & |ab-(a+\Delta_a)(b+\Delta_b)| \\
 & & = & |a\Delta_b+b\Delta_a+\Delta_a\Delta_b| \\
 & & \leq & \Delta(a)\Delta(b)+|a|\Delta(b)+|b|\Delta(a)=\Delta_{dat,\cdot}
\end{array}
$$

$$
\begin{array}{lrcl}
Div.: & |(a/b)-(\tilde{a}/\tilde{b})| & = & \left|\frac{\frac{a}{b}(b+\Delta_b)-(a+\Delta_a)}{b+\Delta_b}\right| = \left|\frac{\frac{a}{b}\Delta_b-\Delta_a}{b+\Delta_b}\right| \\
 & & \leq & \frac{\Delta(a)+|\frac{a}{b}|\Delta(b)}{|b|-\Delta(b)} \\
 & & = & \Delta_{dat,/},\ \text{falls } \Delta(b)<|b|
\end{array}
$$

□

Mit Hilfe von Satz 2.3.2 lassen sich nun gleichmäßige Schranken für die Fortpflanzung des absoluten Datenfehlers für alle $a \in A$ und $b \in B$ herleiten:

Fall 1: Für $|a-\tilde{a}|$ und $|b-\tilde{b}|$ sind die absoluten Schranken $\Delta(a)$ und $\Delta(b)$ gegeben.

$$
\begin{array}{rcl}
\Delta_{dat,+} & = & \Delta(a)+\Delta(b) =: \Delta_{dat}(+) \\
\Delta_{dat,-} & = & \Delta(a)+\Delta(b) =: \Delta_{dat}(-) \\
\Delta_{dat,\cdot} & = & \Delta(a)\Delta(b)+|a|\Delta(b)+|b|\Delta(a) \\
 & \leq & \Delta(a)\Delta(b)+|A|\Delta(b)+|B|\Delta(a) =: \Delta_{dat}(\cdot) \\
\Delta_{dat,/} & = & \frac{\Delta(a)+|\frac{a}{b}|\Delta(b)}{|b|-\Delta(b)} \\
 & \leq & \frac{\Delta(a)+\frac{|A|}{\langle B\rangle}\Delta(b)}{\langle B\rangle-\Delta(b)} =: \Delta_{dat}(/)\text{für}\Delta(b)<\langle B\rangle
\end{array}
$$

Fall 2: Für $|a-\tilde{a}|$ ist die absolute Schranke $\Delta(a)$ und für $|b-\tilde{b}|$ die relative Schranke $\varepsilon(b)$ gegeben.

$$
\Delta_{dat,+} = \Delta(a)+|b|\varepsilon(b) \leq \Delta(a)+|B|\varepsilon(b) =: \Delta_{dat}(+)
$$

$$\begin{aligned}
\Delta_{dat,-} &= \Delta(a) + |b|\varepsilon(b) \;\leq\; \Delta(a) + |B|\varepsilon(b) \;=:\; \Delta_{dat}(-) \\
\Delta_{dat,\cdot} &= \Delta(a)\cdot|b|\varepsilon(b) + |a|\cdot|b|\varepsilon(b) + |b|\Delta(a) \\
&\leq |B|\Big(\Delta(a)\varepsilon(b) + |A|\varepsilon(b) + \Delta(a)\Big) \;=:\; \Delta_{dat}(\cdot) \\
\Delta_{dat,/} &= \frac{\Delta(a) + |\frac{a}{b}|\cdot|b|\varepsilon(b)}{|b| - |b|\varepsilon(b)} \\
&\leq \frac{\Delta(a) + |A|\varepsilon(b)}{\langle B\rangle(1-\varepsilon(b))} \;=:\; \Delta_{dat}(/) \text{ für } \varepsilon(b) < 1
\end{aligned}$$

Fall 3: Für $|a - \tilde{a}|$ ist die relative Schranke $\varepsilon(a)$ und für $|b - \tilde{b}|$ die absolute Schranke $\Delta(b)$ gegeben.

$$\begin{aligned}
\Delta_{dat,+} &= |a|\varepsilon(a) + \Delta(b) \;\leq\; |A|\varepsilon(a) + \Delta(b) \;=:\; \Delta_{dat}(+) \\
\Delta_{dat,-} &= |a|\varepsilon(a) + \Delta(b) \;\leq\; |A|\varepsilon(a) + \Delta(b) \;=:\; \Delta_{dat}(-) \\
\Delta_{dat,\cdot} &= |a|\varepsilon(a)\cdot\Delta(b) + |a|\Delta(b) + |b|\cdot|a|\varepsilon(a) \\
&\leq |A|\Big(\varepsilon(a)\Delta(b) + \Delta(b) + |B|\varepsilon(a)\Big) \;=:\; \Delta_{dat}(\cdot) \\
\Delta_{dat,/} &= \frac{|a|\varepsilon(a) + |\frac{a}{b}|\Delta(b)}{|b| - \Delta(b)} \\
&\leq |A|\frac{\varepsilon(a) + \frac{\Delta(b)}{\langle B\rangle}}{\langle B\rangle - \Delta(b)} \;=:\; \Delta_{dat}(/) \text{ für } \Delta(b) < \langle B\rangle
\end{aligned}$$

Fall 4: Für $|a - \tilde{a}|$ und $|b - \tilde{b}|$ sind die relativen Schranken $\varepsilon(a)$ und $\varepsilon(b)$ gegeben.

$$\begin{aligned}
\Delta_{dat,+} &= |a|\varepsilon(a) + |b|\varepsilon(b) \;\leq\; |A|\varepsilon(a) + |B|\varepsilon(b) \;=:\; \Delta_{dat}(+) \\
\Delta_{dat,-} &= |a|\varepsilon(a) + |b|\varepsilon(b) \;\leq\; |A|\varepsilon(a) + |B|\varepsilon(b) \;=:\; \Delta_{dat}(-) \\
\Delta_{dat,\cdot} &= |a|\varepsilon(a)\cdot|b|\varepsilon(b) + |a|\cdot|b|\varepsilon(b) + |b|\cdot|a|\varepsilon(a) \\
&\leq |A|\cdot|B|\cdot\Big(\varepsilon(a)\varepsilon(b) + \varepsilon(b) + \varepsilon(a)\Big) \;=:\; \Delta_{dat}(\cdot) \\
\Delta_{dat,/} &= \frac{|a|\varepsilon(a) + |\frac{a}{b}|\cdot|b|\varepsilon(b)}{|b| - |b|\varepsilon(b)} \\
&\leq \frac{|A|(\varepsilon(a) + \varepsilon(b))}{\langle B\rangle(1-\varepsilon(b))} \;=:\; \Delta_{dat}(/) \text{ für } \varepsilon(b) < 1
\end{aligned}$$

In Tabelle 2.1 sind die Ergebnisse noch einmal zusammengefaßt.

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

Tabelle 2.1: Schranken für die Fortpflanzung des absoluten Datenfehlers bei den Grundoperationen

2.3.1.2 Abschätzung des Rundungsfehlers

Satz 2.3.3 Für den Rundungsfehler $|(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})|$ einer Operation $\circ \in \{+, -, \cdot, /\}$ mit den Maschinenzahlen $\tilde{a}, \tilde{b} \in S$ gilt

a) im Unterlaufbereich, d. h. $\tilde{a} \boxdot \tilde{b} \in U$ und damit $\tilde{a} \circ \tilde{b} \in U$:

$$|(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})| \;\leq\; \texttt{MinReal} =: \Delta_{rnd,U}(\circ)\,. \tag{2.9}$$

Bei Verwendung einer Arithmetik mit maximal genauen Operationen im „gradual underflow“ kann die Abschätzung noch verschärft werden zu:

$$|(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})| \;\leq\; \left\{ \begin{array}{cc} 0 & \text{für } \circ \in \{+,-\} \\ \texttt{dMinReal} & \text{für } \circ \in \{\cdot, /\} \end{array} \right\} =: \Delta_{rnd,U}(\circ) \tag{2.10}$$

b) im normalisierten Bereich, d. h. $\tilde{a} \boxdot \tilde{b} \notin U$ und damit $\tilde{a} \circ \tilde{b} \notin U$:

$$|(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})| \;\leq\; \bar{\varepsilon} \cdot (\Delta_{dat}(\circ) + |A \circ B|) \;=:\; \Delta_{rnd,N}(\circ) \tag{2.11}$$

für jedes $\circ \in \{+, -, \cdot, /\}$.

Beweis:

a) Ungleichung (2.9) folgt sofort aus Lemma 2.3.1.
Für $\circ \in \{\cdot, /\}$ ist Ungleichung (2.10) offensichtlich
(siehe Bemerkung 2.3).
Seien also $\circ = +$ und $\tilde{a}, \tilde{b} \in S$ mit $\tilde{a} \boxplus \tilde{b} \in U$. Man kann ohne Beschränkung der Allgemeinheit $\tilde{a}, \tilde{b} \geq 0$ annehmen. (Für $\tilde{a}, \tilde{b} \leq 0$ betrachte

$\tilde{a} \boxplus \tilde{b} = -((-\tilde{a}) \boxplus (-\tilde{b}))$. Die beiden anderen Fälle kommen einer Subtraktion gleich, die weiter unten behandelt wird.) Es folgt

$$0 \leq \tilde{a}, \tilde{b} \leq \tilde{a} + \tilde{b} < \texttt{MinReal},$$

d. h. $\tilde{a}$, $\tilde{b}$ und $\tilde{a} + \tilde{b}$ haben denselben Exponenten: $e(\tilde{a}) = e(\tilde{b}) = e(\tilde{a} + \tilde{b}) = e_{min} - 1$. Die Summe $\tilde{a} \boxplus \tilde{b}$ ist daher exakt.
Seien nun $\circ = -$ und $\tilde{a}, \tilde{b} \in S$ mit $\tilde{a} \boxminus \tilde{b} \in U$. Wieder sei o.B.d.A. $\tilde{a}, \tilde{b} \geq 0$ und zusätzlich $\tilde{a} \geq \tilde{b}$. Es sind drei Fälle zu unterscheiden:

Fall I $\quad e(\tilde{a}) = e(\tilde{b})$: klar!

Fall II(a)
$e(\tilde{a}) = e(\tilde{b}) + 1$ und $\tilde{b} \in U$:
Es folgt $\tilde{a} = 1.a_1 \ldots a_{52} \cdot 2^{e_{min}}$ und $\tilde{b} = 0.b_1 \ldots b_{52} \cdot 2^{e_{min}}$, d. h.

$$\begin{array}{rr} & 1.a_1 \ldots a_{52} \cdot 2^{e_{min}} \\ - & 0.b_1 \ldots b_{52} \cdot 2^{e_{min}} \\ \hline = & c_0.c_1 \ldots c_{52} \cdot 2^{e_{min}} \end{array}$$

Da aber $\tilde{a} - \tilde{b} \in U$, ist $c_0 = 0$, und das Ergebnis paßt ohne Rundung in das Gleitpunktsystem.

Fall II(b)
$e(\tilde{a}) = e(\tilde{b}) + 1$ und $\tilde{b} \notin U$:
Es folgt $\tilde{a} = a_0.a_1 \ldots a_{52} \cdot 2^{e(\tilde{a})}$ und $\tilde{b} = 0.b_0 b_1 \ldots b_{52} \cdot 2^{e(\tilde{a})}$ mit $e(\tilde{a}) = e(\tilde{b}) + 1 \geq e_{min} + 1$, d. h.

$$\begin{array}{rr} & a_0.a_1 \ldots a_{52} \quad \cdot 2^{e(\tilde{a})} \\ - & 0.b_0 \ldots b_{51} b_{52} \cdot 2^{e(\tilde{a})} \\ \hline = & c_0.c_1 \ldots c_{52} c_{53} \cdot 2^{e(\tilde{a})} \end{array}$$

Wegen $\tilde{a} - \tilde{b} \in U$ ist $e(\tilde{a} - \tilde{b}) = e_{min} - 1 \leq e(\tilde{a}) - 2$. Es muß also $c_0 = c_1 = 0$ gelten; das Ergebnis paßt somit ohne Rundung in das Gleitpunktsystem.

Fall III
$e(\tilde{a}) \geq e(\tilde{b}) + 2$:
Dann gilt $\tilde{a} \notin U \Longrightarrow \tilde{a} = 1.a_1 \ldots a_{52} \cdot 2^{e(\tilde{a})} \geq 2^{e(\tilde{a})} \geq 2^{e(\tilde{b})+2}$ und $\tilde{b} = b_0.b_1 \ldots b_{52} \cdot 2^{e(\tilde{b})} \leq 2^{e(\tilde{b})+1}$. Daher ist $\tilde{a} - \tilde{b} \geq 2^{e(\tilde{b})+1} \geq 2^{e_{min}} \notin U$, d. h. dieser Fall kann nicht eintreten, wenn die Differenz $\tilde{a} - \tilde{b}$ im Unterlauf liegen soll.

Es folgt in jedem Falle die Behauptung.

b) Weiter gilt

$$\begin{aligned} |(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})| &\leq \bar{\varepsilon} \cdot |\tilde{a} \circ \tilde{b}| \\ &\leq \bar{\varepsilon} \cdot (|(a \circ b) - (\tilde{a} \circ \tilde{b})| + |a \circ b|) \\ &\leq \bar{\varepsilon} \cdot (\Delta_{dat}(\circ) + |A \circ B|) = \Delta_{rnd,N}(\circ)\,, \end{aligned}$$

womit die Behauptung gezeigt wäre.

□

2.3.1.3 Gesamtfehlerabschätzung

Mit der Dreiecksungleichung läßt sich der absolute Gesamtfehler folgendermaßen abschätzen:

$$|(a \circ b) - (\tilde{a} \boxdot \tilde{b})| \leq |(a \circ b) - (\tilde{a} \circ \tilde{b})| + |(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})|\,,$$

d. h. man hat

- im Unterlaufbereich

$$|(a \circ b) - (\tilde{a} \boxdot \tilde{b})| \leq \Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ) =: \Delta(\circ)\,,$$

- im normalisierten Bereich

$$|(a \circ b) - (\tilde{a} \boxdot \tilde{b})| \leq \Delta_{dat}(\circ) + \Delta_{rnd,N}(\circ) =: \Delta(\circ)$$

- und im gesamten Bereich

$$\begin{aligned} |(a \circ b) - (\tilde{a} \boxdot \tilde{b})| &\leq \Delta_{dat}(\circ) + \max(\Delta_{rnd,U}(\circ), \Delta_{rnd,N}(\circ)) \\ &\leq \Delta_{dat}(\circ) + \Delta_{rnd,U}(\circ) + \Delta_{rnd,N}(\circ) =: \Delta(\circ)\,. \end{aligned}$$

2.3.2 Relative Fehlerschranken

Für das Ergebnis der Maschinenverknüpfung $\tilde{a} \boxdot \tilde{b}$ soll nun eine Schranke $\varepsilon(\circ) \in \mathbb{R}^+$ berechnet werden, so daß

$$\left| \frac{(a \circ b) - (\tilde{a} \boxdot \tilde{b})}{a \circ b} \right| \leq \varepsilon(\circ)$$

für alle $a \in A$ mit $|a - \tilde{a}| \leq \Delta(a)$ bzw. $|a - \tilde{a}| \leq \varepsilon(a)|a|$ und für alle $b \in B$ mit $|b - \tilde{b}| \leq \Delta(b)$ bzw. $|b - \tilde{b}| \leq \varepsilon(b)|b|$ gilt.

2.3.2.1 Abschätzung des fortgepflanzten Datenfehlers

Satz 2.3.4 Für die Fortpflanzung des relativen Datenfehlers

$$\left| \frac{(a \circ b) - (\tilde{a} \circ \tilde{b})}{a \circ b} \right|$$

gilt bei exakter Rechnung

$$\left| \frac{(a+b) - (\tilde{a}+\tilde{b})}{a+b} \right| \leq \frac{\Delta(a) + \Delta(b)}{|a+b|} =: \varepsilon_{dat,+} \tag{2.12}$$

$$\left| \frac{(a-b) - (\tilde{a}-\tilde{b})}{a-b} \right| \leq \frac{\Delta(a) + \Delta(b)}{|a-b|} =: \varepsilon_{dat,-} \tag{2.13}$$

$$\left| \frac{(a \cdot b) - (\tilde{a} \cdot \tilde{b})}{a \cdot b} \right| \leq \frac{\Delta(a)}{|a|} \cdot \frac{\Delta(b)}{|b|} + \frac{\Delta(b)}{|b|} + \frac{\Delta(a)}{|a|} =: \varepsilon_{dat,\cdot} \tag{2.14}$$

$$\left| \frac{(a/b) - (\tilde{a}/\tilde{b})}{a/b} \right| \leq \frac{|\frac{b}{a}|\Delta(a) + \Delta(b)}{|b| - \Delta(b)} =: \varepsilon_{dat,/} \text{für } Delta(b) < |b| \,. \tag{2.15}$$

Beweis:

$$\begin{aligned}
Add.: \quad \left| \frac{(a+b) - (\tilde{a}+\tilde{b})}{a+b} \right| &\overset{(2.5)}{\leq} \frac{\Delta(a) + \Delta(b)}{|a+b|} = \varepsilon_{dat,+} \\
Sub.: \quad \left| \frac{(a-b) - (\tilde{a}-\tilde{b})}{a-b} \right| &\overset{(2.6)}{\leq} \frac{\Delta(a) + \Delta(b)}{|a-b|} = \varepsilon_{dat,-} \\
Mul.: \quad \left| \frac{(a \cdot b) - (\tilde{a} \cdot \tilde{b})}{a \cdot b} \right| &\overset{(2.7)}{\leq} \frac{\Delta(a)\Delta(b) + |a|\Delta(b) + |b|\Delta(a)}{|a \cdot b|} \\
&= \frac{\Delta(a)}{|a|} \cdot \frac{\Delta(b)}{|b|} + \frac{\Delta(b)}{|b|} + \frac{\Delta(a)}{|a|} = \varepsilon_{dat,\cdot} \\
Div.: \quad \left| \frac{(a/b) - (\tilde{a}/\tilde{b})}{a/b} \right| &\overset{(2.8)}{\leq} \frac{\Delta(a) + |\frac{a}{b}|\Delta(b)}{|b| - \Delta(b)} \cdot \frac{|b|}{|a|} \\
&= \frac{|\frac{b}{a}|\Delta(a) + \Delta(b)}{|b| - \Delta(b)} \\
&= \varepsilon_{dat,/}, \text{ falls } \Delta(b) < |b|
\end{aligned}$$

□

Mit Hilfe von Satz 2.3.4 lassen sich nun gleichmäßige Schranken für die Fortpflanzung des relativen Datenfehlers für alle $a \in A$ und $b \in B$ herleiten. Für die Addition und Subtraktion werden dabei die beiden folgenden Lemmata verwendet.

Lemma 2.3.5 Seien $x \in X = [\underline{x}, \overline{x}] \in I\!I\!R$, $y \in Y = [\underline{y}, \overline{y}] \in I\!I\!R$ beliebig und $c_1, c_2 \geq 0$ konstant. Dann gilt

$$\max_{x \in X, y \in Y} \frac{c_1|x| + c_2}{|x \pm y|} = \max_{x \in \{\underline{x}, \overline{x}\}, y \in \{\underline{y}, \overline{y}\}} \frac{c_1|x| + c_2}{|x \pm y|} \tag{2.16}$$

$$= \max_{x \in \{\underline{x}, \overline{x}\}} \frac{c_1|x| + c_2}{\langle x \pm Y \rangle} . \tag{2.17}$$

Dabei wird vorausgesetzt, daß die auftretenden Nenner stets ungleich Null sind, d. h. $0 \notin X \pm Y$.

Beweis: Der Beweis wird für den Fall „+“ mit $\inf(X+Y) > 0$ gezeigt. Die restlichen drei Fälle („+“ mit $\sup(X+Y) < 0$, „−“ mit $\inf(X-Y) > 0$ und „−“ mit $\sup(X-Y) < 0$) werden ganz entsprechend behandelt; es ändern sich lediglich einige Vorzeichen.

Sei $D := X \times Y$, $f : D \to I\!R, (x, y) \mapsto f(x, y) := \dfrac{c_1|x| + c_2}{|x + y|} = \dfrac{c_1|x| + c_2}{x + y}$ und $c_1 > 0$ $\left(c_1 = 0 \Longrightarrow \max\limits_{x \in X, y \in Y} \dfrac{c_2}{x + y} = \max\limits_{x \in \{\underline{x}, \overline{x}\}} \dfrac{c_2}{\langle x + Y \rangle} \right)$.
Zu $y_0 \in Y$, beliebig aber fest gewählt, wird $g_{y_0}(x) := f(x, y_0)$ definiert.
Für $x < 0$ ist

$$g'_{y_0}(x) = \frac{-c_1(x + y_0) - (-c_1 x + c_2)}{(x + y_0)^2} = \frac{-c_1 y_0 - c_2}{(x + y_0)^2} < 0 ,$$

da $y_0 > -x > 0$.

Für $x > 0$ ist

$$g'_{y_0}(x) = \frac{c_1(x + y_0) - (c_1 x + c_2)}{(x + y_0)^2} = \frac{c_1 y_0 - c_2}{(x + y_0)^2} \begin{cases} < 0 \text{ für } y_0 < c_2/c_1 , \\ \geq 0 \text{ für } y_0 \geq c_2/c_1 . \end{cases}$$

Somit ist $g_{y_0}(x)$ für $x < 0$ eine monoton fallende und für $x \geq 0$, in Abhängigkeit von der fest vorgegebenen Lage von y_0, eine monoton fallende bzw. monoton wachsende Funktion. Es folgt daher wegen der Stetigkeit von g_{y_0} auf X

$$\max_{x \in X} f(x, y_0) = \max_{x \in X} g_{y_0}(x) = \max_{x \in \{\underline{x}, \overline{x}\}} g_{y_0}(x) \tag{2.18}$$

und weiter

$$\begin{aligned}\max_{x\in X,y\in Y}\frac{c_1|x|+c_2}{x+y} &= \max_{x\in X}\max_{y\in Y}\frac{c_1|x|+c_2}{x+y}\\ &= \max_{x\in X}\max_{y\in\{\underline{y},\overline{y}\}}\frac{c_1|x|+c_2}{x+y}\\ &\overset{(2.18)}{=} \max_{x\in\{\underline{x},\overline{x}\}}\max_{y\in\{\underline{y},\overline{y}\}}\frac{c_1|x|+c_2}{x+y} = \max_{x\in\{\underline{x},\overline{x}\}}\frac{c_1|x|+c_2}{\langle x+Y\rangle}\,.\end{aligned}$$

□

Lemma 2.3.6 Seien $x \in X = [\underline{x},\overline{x}] \in I\!I\!R$, $y \in Y = [\underline{y},\overline{y}] \in I\!I\!R$ beliebig und $d_1, d_2 \geq 0$ konstant. Dann gilt

$$\max_{x\in X,y\in Y}\frac{d_1|x|+d_2|y|}{|x\pm y|} = \max_{x\in\{\underline{x},\overline{x}\},y\in\{\underline{y},\overline{y}\}}\frac{d_1|x|+d_2|y|}{|x\pm y|}\,. \tag{2.19}$$

Dabei wird vorausgesetzt, daß die auftretenden Nenner stets ungleich Null sind, d. h. $0 \notin X \pm Y$.

Beweis: Wieder wird nur der Beweis für den Fall „+“ mit $\inf(X+Y) > 0$ gezeigt. Die anderen Fälle („+“ mit $\sup(X+Y) < 0$, „−“ mit $\inf(X-Y) > 0$ und „−“ mit $\sup(X-Y) < 0$) werden ganz entsprechend gezeigt; es ändern sich lediglich einige Vorzeichen.
Seien o.B.d.A. $d_1, d_2 > 0$ ($d_1 = 0$ oder $d_2 = 0$ führt auf Gleichung (2.17) mit $c_2 = 0$ und evtl. Rollentausch von x und y). Dann folgt die Behauptung aus Lemma 2.3.5:

$$\begin{aligned}\max_{x\in X,y\in Y}\frac{d_1|x|+d_2|y|}{x+y} &= \max_{x\in X}\max_{y\in Y}\frac{d_1|x|+d_2|y|}{x+y}\\ &\overset{2}{=} \max_{x\in X}\max_{y\in\{\underline{y},\overline{y}\}}\frac{d_1|x|+d_2|y|}{x+y}\\ &= \max_{y\in\{\underline{y},\overline{y}\}}\max_{x\in X}\frac{d_1|x|+d_2|y|}{x+y}\\ &\overset{3}{=} \max_{y\in\{\underline{y},\overline{y}\}}\max_{x\in\{\underline{x},\overline{x}\}}\frac{d_1|x|+d_2|y|}{x+y}\,.\end{aligned}$$

□

[2] Lemma 2.3.5 mit $c_1 = d_2$, $c_2 = d_1|x|$
[3] Lemma 2.3.5 mit $c_1 = d_1$, $c_2 = d_2|y|$

Unter Verwendung von Lemma 2.3.5 und Lemma 2.3.6 können jetzt optimale Schranken für die oben genannte Fehlerfortpflanzung angegeben werden.

Fall 1: Für $|a-\tilde{a}|$ und $|b-\tilde{b}|$ sind die absoluten Schranken $\Delta(a)$ und $\Delta(b)$ gegeben.

$$\begin{aligned}
\varepsilon_{dat,+} &= \frac{\Delta(a)+\Delta(b)}{|a+b|} \leq \frac{\Delta(a)+\Delta(b)}{\langle A+B\rangle} =: \varepsilon_{dat}(+)\\
\varepsilon_{dat,-} &= \frac{\Delta(a)+\Delta(b)}{|a-b|} \leq \frac{\Delta(a)+\Delta(b)}{\langle A-B\rangle} =: \varepsilon_{dat}(-)\\
\varepsilon_{dat,\cdot} &= \frac{\Delta(a)}{|a|}\cdot\frac{\Delta(b)}{|b|}+\frac{\Delta(b)}{|b|}+\frac{\Delta(a)}{|a|}\\
&\leq \frac{\Delta(a)}{\langle A\rangle}\cdot\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(a)}{\langle A\rangle} =: \varepsilon_{dat}(\cdot)\\
\varepsilon_{dat,/} &= \frac{|\frac{b}{a}|\Delta(a)+\Delta(b)}{|b|-\Delta(b)} = \frac{\frac{\Delta(a)}{|a|}+\frac{\Delta(b)}{|b|}}{1-\frac{\Delta(b)}{|b|}}\\
&\leq \frac{\frac{\Delta(a)}{\langle A\rangle}+\frac{\Delta(b)}{\langle B\rangle}}{1-\frac{\Delta(b)}{\langle B\rangle}} =: \varepsilon_{dat}(/) \text{ für } \Delta(b)<\langle B\rangle
\end{aligned}$$

Fall 2: Für $|a-\tilde{a}|$ ist die absolute Schranke $\Delta(a)$ und für $|b-\tilde{b}|$ die relative Schranke $\varepsilon(b)$ gegeben.

$$\begin{aligned}
\varepsilon_{dat,+} &= \frac{\Delta(a)+|b|\varepsilon(b)}{|a+b|} \overset{\text{Lemma 2.3.5}}{\leq} \max_{b\in\{\underline{b},\overline{b}\}}\frac{\Delta(a)+|b|\varepsilon(b)}{\langle A+b\rangle} =: \varepsilon_{dat}(+)\\
\varepsilon_{dat,-} &= \frac{\Delta(a)+|b|\varepsilon(b)}{|a-b|} \overset{\text{Lemma 2.3.5}}{\leq} \max_{b\in\{\underline{b},\overline{b}\}}\frac{\Delta(a)+|b|\varepsilon(b)}{\langle A-b\rangle} =: \varepsilon_{dat}(-)\\
\varepsilon_{dat,\cdot} &= \frac{\Delta(a)}{|a|}\cdot\varepsilon(b)+\varepsilon(b)+\frac{\Delta(a)}{|a|}\\
&\leq \frac{\Delta(a)}{\langle A\rangle}\cdot\varepsilon(b)+\varepsilon(b)+\frac{\Delta(a)}{\langle A\rangle} =: \varepsilon_{dat}(\cdot)\\
\varepsilon_{dat,/} &= \frac{|\frac{b}{a}|\Delta(a)+|b|\varepsilon(b)}{|b|-|b|\varepsilon(b)} = \frac{\frac{\Delta(a)}{|a|}+\varepsilon(b)}{1-\varepsilon(b)}
\end{aligned}$$

$$\leq \quad \frac{\frac{\Delta(a)}{\langle A\rangle}+\varepsilon(b)}{1-\varepsilon(b)} \quad =: \ \varepsilon_{dat}(/) \text{ für } \varepsilon(b)<1$$

Fall 3: Für $|a-\tilde{a}|$ ist die relative Schranke $\varepsilon(a)$ und für $|b-\tilde{b}|$ die absolute Schranke $\Delta(b)$ gegeben.

$$\varepsilon_{dat,+} = \frac{|a|\varepsilon(a)+\Delta(b)}{|a+b|} \overset{\text{Lemma 2.3.5}}{\leq} \max_{a\in\{\underline{a},\overline{a}\}} \frac{|a|\varepsilon(a)+\Delta(b)}{\langle a+B\rangle} =: \ \varepsilon_{dat}(+)$$

$$\varepsilon_{dat,-} = \frac{|a|\varepsilon(a)+\Delta(b)}{|a-b|} \overset{\text{Lemma 2.3.5}}{\leq} \max_{a\in\{\underline{a},\overline{a}\}} \frac{|a|\varepsilon(a)+\Delta(b)}{\langle a-B\rangle} =: \ \varepsilon_{dat}(-)$$

$$\varepsilon_{dat,\cdot} = \varepsilon(a)\cdot\frac{\Delta(b)}{|b|}+\frac{\Delta(b)}{|b|}+\varepsilon(a)$$

$$\leq \quad \varepsilon(a)\cdot\frac{\Delta(b)}{\langle B\rangle}+\frac{\Delta(b)}{\langle B\rangle}+\varepsilon(a) \quad =: \ \varepsilon_{dat}(\cdot)$$

$$\varepsilon_{dat,/} = \frac{|\frac{b}{a}|\cdot|a|\varepsilon(a)+\Delta(b)}{|b|-\Delta(b)} = \frac{\varepsilon(a)+\frac{\Delta(b)}{|b|}}{1-\frac{\Delta(b)}{|b|}}$$

$$\leq \quad \frac{\varepsilon(a)+\frac{\Delta(b)}{\langle B\rangle}}{1-\frac{\Delta(b)}{\langle B\rangle}} \quad =: \ \varepsilon_{dat}(/) \text{ für } \Delta(b)<\langle B\rangle$$

Fall 4: Für $|a-\tilde{a}|$ und $|b-\tilde{b}|$ sind die relativen Schranken $\varepsilon(a)$ und $\varepsilon(b)$ gegeben.

$$\varepsilon_{dat,+} = \frac{|a|\varepsilon(a)+|b|\varepsilon(b)}{|a+b|}$$

$$\overset{\text{Lemma 2.3.6}}{\leq} \max_{a\in\{\underline{a},\overline{a}\},b\in\{\underline{b},\overline{b}\}} \frac{|a|\varepsilon(a)+|b|\varepsilon(b)}{|a+b|} =: \ \varepsilon_{dat}(+)$$

$$\varepsilon_{dat,-} = \frac{|a|\varepsilon(a)+|b|\varepsilon(b)}{|a-b|}$$

$$\overset{\text{Lemma 2.3.6}}{\leq} \max_{a\in\{\underline{a},\overline{a}\},b\in\{\underline{b},\overline{b}\}} \frac{|a|\varepsilon(a)+|b|\varepsilon(b)}{|a-b|} =: \ \varepsilon_{dat}(-)$$

$$\varepsilon_{dat,\cdot} = \varepsilon(a)\varepsilon(b)+\varepsilon(b)+\varepsilon(a) \ =: \ \varepsilon_{dat}(\cdot)$$

$$\varepsilon_{dat,/} = \frac{|\frac{b}{a}|\cdot|a|\varepsilon(a)+|b|\varepsilon(b)}{|b|-|b|\varepsilon(b)} = \frac{\varepsilon(a)+\varepsilon(b)}{1-\varepsilon(b)} =: \ \varepsilon_{dat}(/)$$

für $\varepsilon(b) < 1$

In Tabelle 2.2 sind die Ergebnisse noch einmal zusammengefaßt.

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 bei den Grundoperationen

2.3.2.2 Abschätzung des Rundungsfehlers

Über den Rundungsfehler, der bei einer gleitpunktmäßigen Verknüpfung zweier Gleitpunktzahlen entsteht, macht der folgende Satz Aussagen.

Satz 2.3.7 Für den relativen Rundungsfehler

$$\left| \frac{(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})}{a \circ b} \right|$$

einer Operation $\circ \in \{+, -, \cdot, /\}$ mit den Maschinenzahlen $\tilde{a}, \tilde{b} \in S$ gilt

a) im Unterlaufbereich, d. h. $\tilde{a} \boxdot \tilde{b} \in U$ und damit $\tilde{a} \circ \tilde{b} \in U$:

$$\left| \frac{(\tilde{a} \circ \tilde{b}) - (\tilde{a} \boxdot \tilde{b})}{a \circ b} \right| \leq \frac{\texttt{MinReal}}{\langle A \circ B \rangle} =: \varepsilon_{rnd,U}(\circ)\,. \tag{2.20}$$

Bei Verwendung einer Arithmetik mit maximal genauen Operationen im „gradual underflow“ kann die Abschätzung noch verschärft werden

zu:

$$\left|\frac{(\tilde{a}\circ\tilde{b})-(\tilde{a}\boxdot\tilde{b})}{a\circ b}\right| \leq \left\{\begin{array}{ll} 0 & \text{für } \circ\in\{+,-\} \\ \frac{\texttt{dMinReal}}{\langle A\circ B\rangle} & \text{für } \circ\in\{\cdot,/\}\end{array}\right\} =: \varepsilon_{rnd,U}(\circ) \tag{2.21}$$

b) im normalisierten Bereich, d. h. $\tilde{a}\boxdot\tilde{b}\notin U$ und damit $\tilde{a}\circ\tilde{b}\notin U$:

$$\left|\frac{(\tilde{a}\circ\tilde{b})-(\tilde{a}\boxdot\tilde{b})}{a\circ b}\right| \leq \bar{\varepsilon}\cdot(\varepsilon_{dat}(\circ)+1) =: \varepsilon_{rnd,N}(\circ) \tag{2.22}$$

für jedes $\circ\in\{+,-,\cdot,/\}$.

Beweis:

a) Folgt sofort aus Satz 2.3.3a).

b)
$$\begin{aligned}\left|\frac{(\tilde{a}\circ\tilde{b})-(\tilde{a}\boxdot\tilde{b})}{a\circ b}\right| &\leq \bar{\varepsilon}\cdot\left|\frac{\tilde{a}\circ\tilde{b}}{a\circ b}\right| \\ &\leq \bar{\varepsilon}\cdot\frac{|(a\circ b)-(\tilde{a}\circ\tilde{b})|+|a\circ b|}{|a\circ b|} \\ &\leq \bar{\varepsilon}\cdot(\varepsilon_{dat}(\circ)+1)=\varepsilon_{rnd,N}(\circ)\end{aligned}$$

□

2.3.2.3 Gesamtfehlerabschätzung

Mit der Dreiecksungleichung läßt sich der relative Gesamtfehler folgendermaßen abschätzen:

$$\left|\frac{(a\circ b)-(\tilde{a}\boxdot\tilde{b})}{a\circ b}\right| \leq \left|\frac{(a\circ b)-(\tilde{a}\circ\tilde{b})}{a\circ b}\right| + \left|\frac{(\tilde{a}\circ\tilde{b})-(\tilde{a}\boxdot\tilde{b})}{a\circ b}\right|,$$

d. h. man erhält

- im Unterlaufbereich

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

- im normalisierten Bereich

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

- und im gesamten Bereich

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

2.3.3 Berücksichtigung exakt ausführbarer Operationen

In bestimmten Fällen können die in Abschnitt 2.3 hergeleiteten Fehlerschranken verbessert werden. Dies ist immer dann möglich, wenn eine Operation auf dem Rechner ohne Rundungsfehler durchführbar ist. Es ist daher wichtig, Kriterien zu finden, mit deren Hilfe es möglich ist, exakte Operationen zu identifizieren.

Eine generelle Voraussetzung der folgenden Sätze ist, daß die verwendete Gleitkommaarithmetik die Eigenschaft „faithful“ (vgl. Definition A.2.5) besitzt. Für eine Arithmetik, die den IEEE–Standard 754 erfüllt, ist dies jedoch immer der Fall.

Satz 2.3.8 (Subtraktion) Das Ergebnis einer Subtraktion von zwei Maschinenzahlen ist exakt darstellbar, d. h.

$$|(\tilde{a} - \tilde{b}) - (\tilde{a} \boxminus \tilde{b})| = \left| \frac{(\tilde{a} - \tilde{b}) - (\tilde{a} \boxminus \tilde{b})}{a - b} \right| = 0\,,$$

wenn eine der folgenden Bedingungen erfüllt ist:

$$\text{(i)} \qquad \frac{1}{2} \;\leq\; \frac{\tilde{a}}{\tilde{b}} \;\leq\; 2 \tag{2.23}$$

$$\text{(ii)} \qquad e(\tilde{a} - \tilde{b}) \;\leq\; \min(\; e(\tilde{a}) + z_{trail}(\tilde{a}),\; e(\tilde{b}) + z_{trail}(\tilde{b})\;) \tag{2.24}$$

$$\text{(iii)} \qquad \tilde{a} = 0 \text{ oder } \tilde{b} = 0 \tag{2.25}$$

Beweis: Zu (i): Sterbenz [112], zu (ii): Ferguson [37], zu (iii): klar □

Satz 2.3.9 (Addition) Das Ergebnis einer Addition von zwei Maschinenzahlen ist exakt darstellbar, d. h.

$$|(\tilde{a}+\tilde{b})-(\tilde{a}\boxplus\tilde{b})| = \left|\frac{(\tilde{a}+\tilde{b})-(\tilde{a}\boxplus\tilde{b})}{a+b}\right| = 0\,,$$

wenn eine der folgenden Bedingungen erfüllt ist:

$$\text{(i)} \qquad -2 \;\le\; \frac{\tilde{a}}{\tilde{b}} \;\le\; -\frac{1}{2} \tag{2.26}$$

$$\text{(ii)} \qquad e(\tilde{a}+\tilde{b}) \;\le\; \min(\, e(\tilde{a}) + z_{trail}(\tilde{a}),\; e(\tilde{b}) + z_{trail}(\tilde{b})\,) \tag{2.27}$$

$$\text{(iii)} \qquad \tilde{a}=0 \text{ oder } \tilde{b}=0 \tag{2.28}$$

Beweis: Die Behauptung folgt wegen $\tilde{a}+\tilde{b} = \tilde{a}-(-\tilde{b})$ aus Satz 2.3.8. □

Satz 2.3.10 (Multiplikation) Das Ergebnis einer Multiplikation von zwei Maschinenzahlen ist exakt darstellbar, d. h.

$$|(\tilde{a}\cdot\tilde{b})-(\tilde{a}\boxdot\tilde{b})| = \left|\frac{(\tilde{a}\cdot\tilde{b})-(\tilde{a}\boxdot\tilde{b})}{a\cdot b}\right| = 0\,,$$

wenn eine der folgenden Bedingungen erfüllt ist:

$$\text{(i)} \qquad \tilde{a}=2^k,\; k\in\mathbb{Z}. \tag{2.29}$$
Ist $\tilde{a}\cdot\tilde{b}\in U$, muß $k \ge e_{min} - e(\tilde{b}) - z_{trail}(\tilde{b})$ sein.

$$\text{(ii)} \qquad \tilde{b}=2^k,\; k\in\mathbb{Z}. \tag{2.30}$$
Ist $\tilde{a}\cdot\tilde{b}\in U$, muß $k \ge e_{min} - e(\tilde{a}) - z_{trail}(\tilde{a})$ sein.

$$\text{(iii)} \qquad z_{lead}(\tilde{a}) + z_{trail}(\tilde{a}) + z_{lead}(\tilde{b}) + z_{trail}(\tilde{b}) \ge t \tag{2.31}$$

Beweis:

(i) Seien $\tilde{a}=2^k\in S$, $k\in\mathbb{Z}$ und $\tilde{b}\in S$ mit der Darstellung $\tilde{b} = (-1)^{s(\tilde{b})}\cdot m(\tilde{b})\cdot 2^{e(\tilde{b})}$. Dann ist $\tilde{a}\cdot\tilde{b} = 2^k\cdot(-1)^{s(\tilde{b})}\cdot m(\tilde{b})\cdot 2^{e(\tilde{b})} = (-1)^{s(\tilde{b})}\cdot m(\tilde{b})\cdot 2^{k+e(\tilde{b})} \in S$ und daher $\tilde{a}\boxdot\tilde{b} = \tilde{a}\cdot\tilde{b}$.
Für $\tilde{a}\cdot\tilde{b}\in U$ und $k \ge e_{min} - e(\tilde{b}) - z_{trail}(\tilde{b})$ ist $e(\tilde{a}\boxdot\tilde{b}) = e_{min}$, d. h. die Mantisse von $\tilde{b}$ muß bei der Multiplikation mit 2^k um $e_{min}-(k+e(\tilde{b}))$ Stellen nach rechts geschoben werden. Wegen $e_{min}-(k+e(\tilde{b})) \le e_{min}-(e_{min}-e(\tilde{b})-z_{trail}(\tilde{b})+e(\tilde{b})) = z_{trail}(\tilde{b})$ werden nur Nullen aus der Mantisse herausgeschoben, es gilt also auch hier $\tilde{a}\boxdot\tilde{b} = \tilde{a}\cdot\tilde{b}$.

(ii) Analog zu (i).

(iii) $\tilde{a}$ möge $t - (z_{lead}(\tilde{a}) + z_{trail}(\tilde{a}))$ und $\tilde{b}$ möge $t - (z_{lead}(\tilde{b}) + z_{trail}(\tilde{b}))$ signifikante Stellen in der Mantisse besitzen. Dann hat $\tilde{a} \cdot \tilde{b}$ höchstens $t - (z_{lead}(\tilde{a}) + z_{trail}(\tilde{a})) + t - (z_{lead}(\tilde{b}) + z_{trail}(\tilde{b})) = 2t - (z_{lead}(\tilde{a}) + z_{trail}(\tilde{a}) + z_{lead}(\tilde{b}) + z_{trail}(\tilde{b})) \leq 2t - t = t$ signifikante Stellen und damit ist $\tilde{a} \cdot \tilde{b} \in S$.

□

Satz 2.3.11 (Division) Das Ergebnis einer Division von zwei Maschinenzahlen ist exakt darstellbar, d. h.

$$|(\tilde{a}/\tilde{b}) - (\tilde{a} \boxplus \tilde{b})| = \left| \frac{(\tilde{a}/\tilde{b}) - (\tilde{a} \boxplus \tilde{b})}{a/b} \right| = 0\,,$$

wenn die folgende Bedingung erfüllt ist:

$$\tilde{b} = 2^k,\ k \in \mathbb{Z} \tag{2.32}$$

Ist $\tilde{a} \cdot \tilde{b} \in U$, muß zusätzlich $k \leq e(\tilde{a}) + z_{trail}(\tilde{a}) - e_{min}$ gefordert werden.

Beweis: Die Behauptung folgt wegen $\tilde{a}/2^k = \tilde{a} \cdot 2^{-k}$ aus Satz 2.3.10. □

Für die spätere Implementierung werden Bedingungen für $\tilde{A}$ und $\tilde{B}$ benötigt, die gewährleisten, daß sämtliche Operationen $\tilde{a} \circ \tilde{b}$ mit $\tilde{a} \in \tilde{A} \cap S$ und $\tilde{b} \in \tilde{B} \cap S$ rundungsfehlerfrei durchgeführt werden können, also daß

$$\tilde{a} \circ \tilde{b} = \tilde{a} \boxdot \tilde{b} \quad \forall \tilde{a} \in \tilde{A} \cap S\ \ \forall \tilde{b} \in \tilde{B} \cap S$$

gleichmäßig gilt. Diese Bedingungen werden in den folgenden Sätzen angegeben. Dabei sei für $X \in \{\tilde{A}, \tilde{B}\}$

$$e_X := \begin{cases} e(\langle X \rangle), & \text{falls } X \text{ echtes Intervall,} \\ e(x) + z_{trail}(x), & \text{falls } X = [x, x] \text{ Punktintervall.} \end{cases}$$

Satz 2.3.12 (Subtraktion) Ist eine der Bedingungen

$$\text{(i)} \qquad \inf(\tilde{A}/\tilde{B}) \geq \frac{1}{2} \text{ und } \sup(\tilde{A}/\tilde{B}) \leq 2 \tag{2.33}$$

$$\text{(ii)} \qquad e(|\tilde{A} - \tilde{B}|) \leq \min(e_{\tilde{A}}, e_{\tilde{B}}) \tag{2.34}$$

$$\text{(iii)} \qquad \tilde{A} = [0,0] \text{ oder } \tilde{B} = [0,0] \tag{2.35}$$

erfüllt, dann gilt

$$\tilde{a} - \tilde{b} = \tilde{a} \boxminus \tilde{b} \quad \forall \tilde{a} \in \tilde{A} \cap S\ \ \forall \tilde{b} \in \tilde{B} \cap S\,.$$

Beweis:

(i) $\inf(\tilde{A}/\tilde{B}) \geq \frac{1}{2} \ \wedge \ \sup(\tilde{A}/\tilde{B}) \leq 2 \implies \frac{1}{2} \leq \frac{\tilde{a}}{\tilde{b}} \leq 2 \quad \forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S \overset{\text{Satz 2.3.8 (i)}}{\Longrightarrow}$ Beh.

(ii) $e(\tilde{a} - \tilde{b}) \leq e(|\tilde{A} - \tilde{B}|)$
$\leq \min(e_{\tilde{A}}, e_{\tilde{B}}) \leq \min(\ e(\tilde{a}) + z_{trail}(\tilde{a}),\ e(\tilde{b}) + z_{trail}(\tilde{b})\ )$
$\forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S \overset{\text{Satz 2.3.8 (ii)}}{\Longrightarrow}$ Beh.

(iii) Folgt aus Satz 2.3.8 (iii).

□

Satz 2.3.13 (Addition) Ist eine der Bedingungen

$$\text{(i)} \quad \inf(\tilde{A}/\tilde{B}) \ \geq \ -2 \text{ und } \sup(\tilde{A}/\tilde{B}) \ \leq \ -\frac{1}{2} \tag{2.36}$$

$$\text{(ii)} \quad e(|\tilde{A} + \tilde{B}|) \ \leq \ \min(e_{\tilde{A}}, e_{\tilde{B}}) \tag{2.37}$$

$$\text{(iii)} \quad \tilde{A} = [0,0] \text{ oder } \tilde{B} = [0,0] \tag{2.38}$$

erfüllt, dann gilt

$$\tilde{a} + \tilde{b} = \tilde{a} \boxplus \tilde{b} \quad \forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S\,.$$

Beweis:

(i) $\inf(\tilde{A}/\tilde{B}) \geq -2 \ \wedge \ \sup(\tilde{A}/\tilde{B}) \leq -\frac{1}{2} \implies -2 \leq \frac{\tilde{a}}{\tilde{b}} \leq -\frac{1}{2} \quad \forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S \overset{\text{Satz 2.3.9 (i)}}{\Longrightarrow}$ Beh.

(ii) $e(\tilde{a} + \tilde{b}) \leq e(|\tilde{A} + \tilde{B}|)$
$\leq \min(e_{\tilde{A}}, e_{\tilde{B}}) \leq \min(\ e(\tilde{a}) + z_{trail}(\tilde{a}),\ e(\tilde{b}) + z_{trail}(\tilde{b})\ )$
$\forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S \overset{\text{Satz 2.3.9 (ii)}}{\Longrightarrow}$ Beh.

(iii) Folgt aus Satz 2.3.9 (iii).

□

Satz 2.3.14 (Multiplikation) Ist eine der Bedingungen

$$\text{(i)} \quad \begin{array}{l} \tilde{A} = [\tilde{a}, \tilde{a}],\ \tilde{a} = 2^k,\ k \in \mathbb{Z}; \\ \text{ist } (\tilde{A} \cdot \tilde{B}) \cap U \neq \emptyset,\ \text{muß } k \geq e_{min} - e_{\tilde{B}} \text{ sein} \end{array} \tag{2.39}$$

(ii) $\tilde{B} = [\tilde{b}, \tilde{b}],\ \tilde{b} = 2^k,\ k \in \mathbb{Z}$; (2.40)
ist $(\tilde{A} \cdot \tilde{B}) \cap U \neq \emptyset$, muß $k \geq e_{min} - e_{\tilde{A}}$ sein

(iii) $\tilde{A} = [\tilde{a}, \tilde{a}],\ \tilde{B} = [\tilde{b}, \tilde{b}]$, (2.41)
$z_{lead}(\tilde{a}) + z_{trail}(\tilde{a}) + z_{lead}(\tilde{b}) + z_{trail}(\tilde{b}) \geq t$ (2.42)

erfüllt, dann gilt

$$\tilde{a} \cdot \tilde{b} = \tilde{a} \boxdot \tilde{b} \quad \forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S\,.$$

Beweis: Die Behauptung folgt leicht aus Satz 2.3.10 (i) bis (iii). □

Satz 2.3.15 (Division) Es gilt

$$\tilde{a} / \tilde{b} = \tilde{a} \boxslash \tilde{b} \quad \forall \tilde{a} \in \tilde{A} \cap S \ \ \forall \tilde{b} \in \tilde{B} \cap S\,,$$

falls die folgende Bedingung erfüllt ist:

$$\tilde{B} = [\tilde{b}, \tilde{b}],\ \tilde{b} = 2^k,\ k \in \mathbb{Z}. \tag{2.43}$$

Ist $(\tilde{A} \cdot \tilde{B}) \cap U \neq \emptyset$, muß zusätzlich $k \leq e_{\tilde{A}} - e_{min}$ gefordert werden.

Beweis: Die Behauptung folgt leicht aus Satz 2.3.11. □

Bemerkung: Die Aussagen über den Rundungsfehler im Unterlaufbereich sind natürlich nur zulässig, wenn die zugrundeliegende Arithmetik den „gradual underflow“ auch unterstützt!

Bemerkung: Ob für $\tilde{A} = [\tilde{a}, \tilde{a}],\ \tilde{B} = [\tilde{b}, \tilde{b}]$ die Operation $\tilde{a} \circ \tilde{b}$ rundungsfehlerfrei durchführbar ist, läßt sich im Falle einer maximal genauen Maschinen-Intervallarithmetik auch leicht mit der entsprechenden Maschinen-Intervalloperation prüfen. Es gilt nämlich

$$d(\tilde{A} \diamondsuit \tilde{B}) = 0 \implies \tilde{a} \circ \tilde{b} = \tilde{a} \boxdot \tilde{b}. \tag{2.44}$$

Bemerkung: Ist die verwendete Maschinen-Intervallarithmetik maximal genau und wird zusätzlich zu (2.33) bis (2.43) $d(\tilde{A} \diamondsuit \tilde{B}) = 0$ geprüft, kann (2.42) entfallen.

2.4 Fehlerschranken für Funktionen

Satz 2.4.1 Sei $f : D \to I\!R$ differenzierbar auf dem Intervall

$$\tilde{A} = A + [-\Delta(a), \Delta(a)] \subset I\!R\,.$$

Für die auf der Maschine implementierte Näherungsfunktion $\tilde{f} : S \to S$ an f sei $\varepsilon(f) \geq 0$ eine obere Schranke für den relativen Fehler im normalisierten Bereich und $\Delta(f) \geq 0$ eine obere Schranke für den absoluten Fehler im Unterlaufbereich, d. h. es gelte

$$|f(\tilde{x}) - \tilde{f}(\tilde{x})| \leq \varepsilon(f)|f(\tilde{x})| \tag{2.45}$$

für alle $\tilde{x} \in S$ mit $|f(\tilde{x})| \in [\texttt{MinReal}, \texttt{MaxReal}]$ und

$$|f(\tilde{x}) - \tilde{f}(\tilde{x})| \leq \Delta(f) \tag{2.46}$$

für alle $\tilde{x} \in S$ mit $|f(\tilde{x})| \in [0, \texttt{MinReal})$. Mit diesen Voraussetzungen erhält man die Fehlerabschätzung

$$|f(a) - \tilde{f}(\tilde{a})| \leq \Delta_{dat,f} + \varepsilon(f)\Big(\Delta_{dat,f} + |f(a)|\Big) + \Delta(f) \;\; \forall a \in A\,, \tag{2.47}$$

wobei

$$\begin{aligned} \Delta_{dat,f} &= \Delta(a) \cdot \Big|f'\Big(a + [-\Delta(a), \Delta(a)]\Big)\Big| \\ &= |a|\varepsilon(a) \cdot \Big|f'\Big(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\Big)\Big|\,. \end{aligned} \tag{2.48}$$

Beweis: Seien $a \in A$ und $\tilde{a} \in \tilde{A} \cap S$ beliebig und $\Delta_a = \tilde{a} - a \in [-\Delta(a), \Delta(a)]$.

1. **fortgepflanzter Datenfehler:**
   Aus dem Mittelwertsatz der Differentialrechnung folgt

   $$f(\tilde{a}) = f(a + \Delta_a) = f(a) + f'(a + \theta\Delta_a)\Delta_a \text{ mit } \theta \in (0\,, 1)\,,$$

   d. h. für die Fortpflanzung des Datenfehlers Δ_a gilt bei exakter Rechnung

   $$\begin{aligned} |f(a) - f(\tilde{a})| &= |\Delta_a f'(a + \theta\Delta_a)| \\ &\leq \Delta(a) \cdot \Big|f'\Big(a + [-\Delta(a), \Delta(a)]\Big)\Big| \\ &= |a|\varepsilon(a) \cdot \Big|f'\Big(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\Big)\Big| = \Delta_{dat,f}\,. \end{aligned}$$

2. **Rundungsfehler:**
Nach (2.45) und (2.46) ist

$$|f(\tilde{a}) - \tilde{f}(\tilde{a})| \leq \begin{cases} \Delta(f)\,, & \text{falls } |f(\tilde{a})| \in [0, \texttt{MinReal})\,, \\ \varepsilon(f)|f(\tilde{a})|\,, & \text{falls } |f(\tilde{a})| \in [\texttt{MinReal}, \texttt{MaxReal}]\,. \end{cases}$$

Man hat also

$$\begin{aligned} |f(\tilde{a}) - \tilde{f}(\tilde{a})| &\leq \varepsilon(f)|f(\tilde{a})| + \Delta(f) \\ &\leq \varepsilon(f)\Big(|f(\tilde{a}) - f(a)| + |f(a)|\Big) + \Delta(f) \\ &= \varepsilon(f)\Big(\Delta_{dat,f} + |f(a)|\Big) + \Delta(f)\,. \end{aligned}$$

3. **Gesamtfehler:**
Da $a \in A$ und $\tilde{a} \in \tilde{A} \cap S$ beliebig waren, folgt die Behauptung jetzt leicht aus der Dreiecksungleichung, angewandt auf den Gesamtfehler:

$$\begin{aligned} |f(a) - \tilde{f}(\tilde{a})| &\leq |f(a) - f(\tilde{a})| + |f(\tilde{a}) - \tilde{f}(\tilde{a})| \\ &\leq \Delta_{dat,f} + \varepsilon(f)\Big(\Delta_{dat,f} + |f(a)|\Big) + \Delta(f) \quad \forall a \in A\,. \end{aligned}$$

□

Korollar 2.4.2 Für den relativen Gesamtfehler gilt nach Satz 2.4.1

$$\left|\frac{f(a) - \tilde{f}(\tilde{a})}{f(a)}\right| \leq \varepsilon_{dat,f} + \varepsilon(f)\Big(\varepsilon_{dat,f} + 1\Big) + \frac{\Delta(f)}{|f(a)|} \quad \forall a \in A\,, \tag{2.49}$$

wobei

$$\begin{aligned} \varepsilon_{dat,f} &= \Delta(a) \cdot \left|\frac{f'\Big(a + [-\Delta(a), \Delta(a)]\Big)}{f(a)}\right| \\ &= |a|\varepsilon(a) \cdot \left|\frac{f'\Big(a \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\Big)}{f(a)}\right|\,. \end{aligned} \tag{2.50}$$

Korollar 2.4.3 Gleichmäßige Fehlerabschätzungen über dem Intervall A erhält man, indem man (2.47) und (2.49) intervallmäßig für A auswertet,

also

$$\begin{aligned} |f(a) - \tilde{f}(\tilde{a})| &\leq \Delta_{dat}(f) + \varepsilon(f)\Big(\Delta_{dat}(f) + |f(A)|\Big) \\ &\quad + \Delta(f) \;\; \forall a \in A \end{aligned} \tag{2.51}$$

$$\begin{aligned} \left|\frac{f(a) - \tilde{f}(\tilde{a})}{f(a)}\right| &\leq \varepsilon_{dat}(f) + \varepsilon(f)\Big(\varepsilon_{dat}(f) + 1\Big) \\ &\quad + \frac{\Delta(f)}{\langle f(A)\rangle} \;\; \forall a \in A\,, \end{aligned} \tag{2.52}$$

wobei

$$\begin{aligned} \Delta_{dat}(f) &= \Delta(a) \cdot \Big|f'\Big(A + [-\Delta(a), \Delta(a)]\Big)\Big| \\ &= |A|\varepsilon(a) \cdot \Big|f'\Big(A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\Big)\Big| \end{aligned} \tag{2.53}$$

und

$$\begin{aligned} \varepsilon_{dat}(f) &= \Delta(a) \cdot \frac{\Big|f'\Big(A + [-\Delta(a), \Delta(a)]\Big)\Big|}{\langle f(A)\rangle} \\ &= |A|\varepsilon(a) \cdot \frac{\Big|f'\Big(A \cdot [1 - \varepsilon(a), 1 + \varepsilon(a)]\Big)\Big|}{\langle f(A)\rangle}\,. \end{aligned} \tag{2.54}$$

Bemerkung: Manchmal wird von der Approximation $\tilde{f}$ verlangt, daß sie dieselben Nullstellen wie f hat, also

$$f(\tilde{x}) = 0 \implies \tilde{f}(\tilde{x}) = 0\,.$$

Die oben aufgeführten Fehlerschranken behalten jedoch ihre Gültigkeit auch dann, wenn diese Eigenschaft nicht erfüllt ist.

Bemerkung: Die Ableitung von f in (2.53) bzw. (2.54) kann mit Hilfe von automatischer Differentiation leicht berechnet werden. Treten in der intervallmäßigen Auswertung von (2.51) bzw. (2.52) zu große Überschätzungen auf, können die Fehlerschranken durch direkte Anwendung von Satz 2.4.1 und Korollar 2.4.2 eventuell verschärft werden[4].

Die Herleitung konkreter Fehlerschranken für die Funktionen $\sqrt{x}$, e^x und $\ln(x)$ findet sich in Bantle [11].

[4] Gelegentlich kann gezeigt werden, daß die Fehlerschranken ihr Maximum am Intervallrand annehmen.

2.5 Implementierung

Die hergeleiteten Fehlerabschätzungen können nun mit Hilfe von Intervallrechnung in Intervallroutinen umgesetzt werden. Zur Implementierung wird die Programmiersprache Pascal–XSC verwendet. Die Umsetzung erfolgt mit Hilfe des Modulkonzepts von Pascal–XSC in drei verschiedenen Modulen:

Modulname	Art des Fehlerkalküls
`abs_ari`	Kalkül zur Berechnung absoluter Schranken
`rel_ari`	Kalkül zur Berechnung relativer Schranken
`bt_ari`	kombinierter Kalkül für abs. und rel. Schranken

Der Verbunddatentyp `BoundType` ist je nach verwendetem Modul (`abs_ari`, `rel_ari` oder `bt_ari`) unterschiedlich aufgebaut:

```
BoundType = record
               Enclosure: interval;
               AbsErr: real;
            end;
```

oder

```
BoundType = record
               Enclosure: interval;
               RelErr: real;
            end;
```

oder

```
Select = (Abs,Rel);
BoundType = record
               Enclosure: interval;
               Err: real;
               Select: flag;
            end;
```

Durch Ausnutzung des von Pascal–XSC zur Verfügung gestellten Konzepts der Operatorüberladung können die implementierten Routinen vom Benutzer sehr einfach und elegant verwendet werden. Im Anhang B.1 sind einige Routinen der Implementierung in der Programmiersprache Pascal–XSC exemplarisch abgedruckt.

Eine zwischenzeitlich erfolgte Umsetzung des Kalküls nach C–XSC findet sich bei Bantle [11].

2.6 Anwendungsbeispiel

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

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

Mit dem oben vorgestellten Fehlerkalkül erhält man folgende sichere Fehlerschranken:

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

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

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

Kapitel 3

Approximationsfehlerabschätzungen

In diesem Kapitel wird für Approximationsfehlerabschätzungen eine Methode vorgestellt, die von Krämer [68] entwickelt wurde. Eine ausführliche Beschreibung mit Anwendung auf die Fehler– und die komplementäre Fehlerfunktion findet man bei Blomquist/Krämer [18]. Die hier gegebene Beschreibung lehnt sich an die zitierte Arbeit an.

3.1 Grundlagen zur Fehlerabschätzung bei speziellen Funktionen

Im Zusammenhang mit der Entwicklung und Implementierung spezieller Funktionen der mathematischen Physik zeigt sich, daß es für die sichere Fehlerabschätzung oft sinnvoll ist, zusätzlich zur eigentlich gesuchten, auf dem Rechner zu realisierenden Näherung $g(x)$, eine Hilfsnäherungsfunktion $H(x)$ zu betrachten. Diese Funktion $H(x)$ ist in der Regel eine fast perfekte Approximation, die z. B. mit einer Langzahlarithmetik berechnet wird.

Sei $f : [a,b] \longrightarrow \mathbb{R}$ eine stetige reellwertige Funktion die auf einem Rechner mit Gleitkommaraster $S(B,k)$ ausgewertet werden soll. Um kurze Laufzeiten zu erhalten, soll die Approximationsfunktion $g \approx f$ möglichst einfach (z. B. als rationale Funktion) aufgebaut sein. Bezeichnet man mit $\widetilde{f}(x)$ das i.a. fehlerbehaftete Maschinenergebnis, so gilt

$$\widetilde{f}(x) = f(x)(1+\varepsilon_f), \qquad |\varepsilon_f| \le \varepsilon(f) \quad \text{für alle } x \in [a,b] \cap S(B,k).$$

Zur Berechnung der gesuchten Fehlerschranke $\varepsilon(f)$ sind i.a. zwei Approximationsschritte notwendig. Zunächst muß eine Hilfsfunktion $H(x)$ als Approximation an $f(x)$ gefunden werden, welche durchaus recht kompliziert aufgebaut sein darf. Es wird nur verlangt, daß $H(x)$ mit Hilfe von bereits implementierten Intervallfunktionen bzw. Intervalloperationen programmierbar ist. Die zugehörige Approximationsfehlerschranke $\varepsilon(\text{app},1)$ muß in der Regel analytisch (per Hand) hergeleitet werden. Die Gewinnung der eigentlichen Approximationsfunktion $g(x)$ für die Implementierung der Ausgangsfunktion $f(x)$ geschieht dann mit Hilfe von $H(x)$. Eine Schranke $\varepsilon(\text{app},2)$ für den hierbei auftretenden (zweiten) Approximationsfehler kann nun mit intervallarithmetischen Mitteln automatisch bestimmt werden. Genauer wird wie folgt vorgegangen:

Schritt 1: $f(x) \approx H(x), \qquad x \in [a,b],$
wobei $H(x)$ nur aus den Standardfunktionen aufgebaut sein soll, die das Intervall-Langzahlmodul `mpitaylor` zur Verfügung stellt. Der relative Approximationsfehler ist für $f(x) \neq 0$ gegeben durch:

$$\varepsilon_{\text{app},1} = \frac{f(x) - H(x)}{f(x)}; \qquad |\varepsilon_{\text{app},1}| \leq \varepsilon(\text{app},1), \quad x \in [a,b].$$

Die Berechnung von $\varepsilon(\text{app},1)$ ist somit ein rein mathematisches Problem, das für jede Funktion individuell zu lösen ist. Dabei wird nicht verlangt, daß $H(x)$ auf dem Rechner möglichst schnell auszuwerten ist; das Hauptziel ist vielmehr die Berechnung einer garantierten Oberschranke $\varepsilon(\text{app},1)$ des entsprechenden Approximationsfehlers!

Schritt 2: $H(x) \approx g(x), \qquad x \in [a,b],$

wobei g jetzt diejenige Funktion bezeichnet, mit der f auf dem Rechner tatsächlich approximiert wird. Um kurze Laufzeiten zu erhalten, wird g in vielen Fällen als gebrochen rationale Funktion, deren Koeffizienten z.B. mit Hilfe eines Computeralgebrasystems berechnet werden können, gewählt. Für den relativen Approximationsfehler (bezüglich der Hilfsfunktion $H(x)$) kann man unter der Annahme $H(x) \neq 0$ die Oberschranke $\varepsilon(\text{app},2)$ automatisch mit dem Rechner bestimmen:

$$\varepsilon_{\text{app},2} = \frac{H(x) - g(x)}{H(x)}, \qquad |\varepsilon_{\text{app},2}| \leq \varepsilon(\text{app},2), \quad x \in [a,b].$$

Dies kann mit Hilfe eines Pascal–XSC Programms bestimmt werden (Krämer [68]).

Bezüglich der Näherung $f \approx g$ ist der relative Approximationsfehler definiert durch

$$\varepsilon_{\text{app}} := \frac{f(x) - g(x)}{f(x)}, \qquad |\varepsilon_{\text{app}}| \leq \varepsilon(\text{app}), \quad x \in [a,b]$$

und $|\varepsilon_{\text{app}}|$ läßt sich mit Hilfe der Schranken $\varepsilon(\text{app},1)$ und $\varepsilon(\text{app},2)$ durch Anwendung der Dreiecksungleichung abschätzen:

$$|\varepsilon_{\text{app}}| \leq \varepsilon(\text{app},1) + [1 + \varepsilon(\text{app},1)] \cdot \varepsilon(\text{app},2) =: \varepsilon(\text{app})\,. \tag{3.1}$$

Bezeichnet man mit $\widetilde{g}(x)$ das i.a. fehlerbehaftete Rechnerergebnis von g, so ergibt sich die Darstellung

$$\widetilde{g}(x) = g \cdot (1 + \varepsilon_g)\,,$$

und die Oberschranke $\varepsilon(g)$ für den maximalen Betrag des relativen Auswertefehlers ε_g läßt sich mit Hilfe des Fehlerkalküls (Kapitel 2) automatisch berechnen. Danach gilt dann

$$\widetilde{g}(x) = g(x) \cdot (1 + \varepsilon_g); \qquad |\varepsilon_g| \leq \varepsilon(g) \quad \text{für alle } x \in [a,b] \cap S(B,k).$$

Mit der Gesamtfehlerabschätzung erhält man schließlich mit $\widetilde{f}(x) = f(x)(1+\varepsilon_f)$

$$|\varepsilon_f| \leq \varepsilon(\text{app}) + [1 + \varepsilon(\text{app})] \cdot \varepsilon(g) =: \varepsilon(f)\,. \tag{3.2}$$

Vorausgesetzt ist dabei

$$x \in [a,b] \cap S(B,k) \;\wedge\; f = 0 \quad \Longrightarrow \quad \widetilde{f}(x) \equiv \widetilde{g}(x) = 0\,.$$

Diese Forderung kann durch eine geeignete Sonderbehandlung der Nullstellen der betrachteten Funktion bei der Entwicklung des Algorithmus für g in der Regel einfach erfüllt werden.

3.2 Approximationsfehler

3.2.1 Generelles Vorgehen

Bei der Implementierung einer über einem reellen Intervall definierten reellwertigen Funktion

$$f : [a,b] \longrightarrow I\!R$$

auf einem Rechner sind die folgenden zwei Punkte zu beachten:

- Um möglichst kurze Laufzeiten zu erhalten, sollte f über dem Intervall $[a,b]$ durch eine rationale Funktion approximiert werden, wobei Zähler und Nenner durch je ein Polynom definiert sind:

$$f(x) \approx g(x) := \frac{\mathrm{P}_N(x)}{\mathrm{Q}_M(x)};\ \mathrm{Q}_M(x) \neq 0,\ x \in [a,b],\ N,M \in \{0,1,2,...\}$$

- Wegen der Approximation $f(x) \approx g(x)$ und wegen der i.a. unvermeidbaren Rundungsfehler bei der Auswertung von g wird das Maschinenergebnis mit dem exakten Funktionswert $f(x)$ nur in Ausnahmefällen übereinstimmen. Zur Berechnung einer garantierten Fehlerschranke ist es daher u.a. notwendig, eine ebenfalls garantierte Oberschranke für den Approximationsfehler für alle $x \in [a,b]$ zu bestimmen.

Der absolute bzw. relative Approximationsfehler ist definiert durch

$$\begin{aligned} \Delta(x) &:= f(x) - g(x), \quad |\Delta(x)| \leq \Delta(\mathrm{app}), \quad x \in [a,b], \\ \varepsilon(x) &:= \frac{f(x) - g(x)}{f(x)}, \quad |\varepsilon(x)| \leq \varepsilon(\mathrm{app}), \quad f(x) \neq 0, \quad x \in [a,b]. \end{aligned}$$

Zur Bestimmung der Oberschranken $\Delta(\mathrm{app}), \varepsilon(\mathrm{app})$ gibt es in Abhängigkeit von der vorgegebenen Funktion f und ihrem Approximationsintervall $[a,b]$ verschiedene Methoden.

Bei den elementaren Funktionen ($f = \exp, \sin, \arctan, ...$) läßt sich $[a,b]$ auf ein relativ schmales Intervall reduzieren, dessen Mittelpunkt der Koordinatenursprung ist. Als Approximationsfunktion kann daher ein Taylor-Polynom niedriger Ordnung (kurze Laufzeit) gewählt werden, und der Rest der Taylorreihe läßt sich entweder durch eine geometrische Reihe oder durch das nachfolgende Reihenglied abschätzen, wenn die Potenzreihe eine Leibnizreihe ist (Braune [24], Krämer [58]). Dadurch erhält man für die Oberschranke des Approximationsfehlers einen in geschlossener Form vorliegenden einfachen Ausdruck, der durch Intervallrechnung sicher nach oben abgeschätzt werden kann.

Bei der Berechnung des Approximationsfehlers für die speziellen Funktionen der mathematischen Physik liegen die Dinge etwas anders, da jetzt im Gegensatz zu den Standardfunktionen eine Argumentreduktion auf ein sehr schmales Intervall i. a. nicht möglich ist.

Das folgende Beispiel soll den Sachverhalt verdeutlichen. Mit der Riemannschen Zeta-Funktion $\zeta(x)$ und der Eulerschen Konstanten $\gamma =$

$0.57721\ldots$ gilt für die Funktion $f(x) = -\ln(\Gamma(x))$ die Reihenentwicklung

$$f(x) \equiv (x-2)(\gamma-1) - \sum_{k=2}^{\infty}(-1)^k[\zeta(k)-1]\cdot\frac{(x-2)^k}{k}, \quad |x-2| \le \frac{1}{2}.$$

Approximiert man nun $f(x)$ im Intervall $[1.5, 2.5]$ durch das Taylorpolynom

$$\mathrm{T}_N(x) := (x-2)(\gamma-1) - \sum_{k=2}^{N}(-1)^k[\zeta(k)-1]\cdot\frac{(x-2)^k}{k} \approx f(x),$$

so erhält man nach dem oben beschriebenen Verfahren erst mit N=26 für den absoluten Fehler die Oberschranke $\Delta(\mathrm{app}) = 4.1121\cdot 10^{-18}$. Aus Laufzeitgründen ist die Rechnerauswertung von $\mathrm{T}_{26}(x)$ also völlig unakzeptabel. Die Rechenzeit reduziert sich jedoch etwa um den Faktor 2.4, wenn man $\mathrm{T}_{26}(x)$ durch eine rationale Bestapproximation ersetzt

$$f(x) \approx \mathrm{T}_{26}(x) \approx \frac{\mathrm{P}_6(x-2)}{\mathrm{Q}_5(x-2)}, \quad |x-2| \le \frac{1}{2} \tag{3.3}$$

mit

$$\mathrm{P}_6(x-2) := \sum_{k=0}^{6} a_k\cdot(x-2)^k, \qquad \mathrm{Q}_5(x-2) := \sum_{k=0}^{5} b_k\cdot(x-2)^k\,.$$

Die Polynomkoeffizienten a_k, b_k können dabei z.B. mit einem Computeralgebrasystem (Langzahlrechnung) bestimmt werden. Für die rationale Approximation muß jetzt der Approximationsfehler bzgl. T_{26} sicher abgeschätzt werden.
Um dieses Problem etwas allgemeiner zu formulieren, ersetzt man das spezielle Polynom $\mathrm{T}_{26}(x)$ durch eine hinreichend oft differenzierbare Hilfsfunktion $H(x)$, welche im Bereich $|x-x_0| \le \eta$ nur aus endlich vielen Standardfunktionen aufgebaut sein soll, die im XSC-Modul `mpitaylor` bereitgestellt werden. Man sucht also für die folgende Approximation

$$H(x) \approx \frac{\mathrm{P}_N(x-x_0)}{\mathrm{Q}_M(x-x_0)}, \quad \mathrm{Q}_M(x-x_0) \neq 0, \quad |x-x_0| \le \eta \tag{3.4}$$

die Oberschranken $\Delta(\mathrm{app})$ bzw. $\varepsilon(\mathrm{app})$ des absoluten bzw. relativen Approximationsfehlers. Für $\varepsilon(x)$ ergibt sich z. B. die Darstellung

$$\varepsilon(x) := \frac{\mathrm{P}_N(x-x_0) - \mathrm{Q}_M(x-x_0)\cdot H(x)}{\mathrm{Q}_M(x-x_0)\cdot H(x)}, \quad |x-x_0| \le \eta, \ \text{Nenner} \neq 0, \tag{3.5}$$

und die Berechnung einer garantierten Oberschranke $\varepsilon(\mathrm{app})$ für $|\varepsilon(x)|$ erscheint auf den ersten Blick einfach, da man gewohnt ist, ähnliche Probleme mit Werkzeugen der (verifizierten) globalen Optimierung ohne Schwierigkeiten zu lösen. Es stellt sich heraus, daß $\varepsilon(\mathrm{app})$ durch globale Optimierung grundsätzlich nicht bestimmt werden kann! Um dies einzusehen, sei zunächst daran erinnert, daß die globale Optimierung nur dann zum Ziel führt, wenn man die Wertebereiche von $\varepsilon(x)$ für jedes Teilintervall einer endlichen Zerlegung von $|x - x_0| \leq \eta$ ohne wesentliche Überschätzung berechnen kann. Bedeutet $[x]$ ein solches Teilintervall, so ist im Zähler von (3.5) der Ausdruck

$$\mathrm{P}_N([x] - x_0) - \mathrm{Q}_M([x] - x_0) \cdot H([x]) \tag{3.6}$$

intervallmäßig auszuwerten, wobei die Ergebnisintervalle von Minuend und Subtrahend um so besser übereinstimmen, je höher bei der rationalen Approximation die Polynomgrade N, M gewählt werden. Im Ausdruck (3.6) sind damit zwei fast identische und nicht punktförmige Intervalle zu subtrahieren, was bekanntlich zu sehr starken Überschätzungen führt. Dies ist der Grund für das Versagen der globalen Optimierungsalgorithmen beim Untersuchen von Fehlerkurven bei Approximationsproblemen. Das Problem wird übrigens auch nicht dadurch gelöst, daß man in (3.6) die beiden Intervall-Summanden mit dem Intervall-Langzahlmodul `mpi_ari` in hoher Genauigkeit auswertet, denn die Tatsache, daß zwei fast identische Intervalle zu subtrahieren sind, wird auch durch eine Langzahlarithmetik nicht beseitigt. Eine rein theoretische Lösung würde darin bestehen, daß man die Teilintervalle $[x]$ quasi punktförmig wählt, was jedoch auf völlig unpraktikable Rechenzeiten führen würde.

Eine Abschätzung des Approximationsfehlers wird also nur möglich sein, wenn es gelingt, den Ausdruck (3.6) so umzuformen, daß die Subtraktion fast identischer Intervalle vermieden wird. Dazu entwickeln man $H(x)$ im Punkt x_0 z. B. mit den Methoden der automatischen Differentiation in ein Taylor-Polynom mit Restglied $H(x) = \sum_{k=0}^{K} s_k \cdot (x - x_0)^k + \mathrm{R}(x, K)$. Man erhält für den Ausdruck (3.6) die Darstellung

$$\underbrace{\left[\mathrm{P}_N(x - x_0) - \mathrm{Q}_M(x - x_0) \cdot \sum_{k=0}^{K} s_k \cdot (x - x_0)^k\right]}_{(*)} - \mathrm{Q}_M(x - x_0) \cdot \mathrm{R}(x, K)\,. \tag{3.7}$$

Der eigentliche Trick besteht nun darin, die Polynomdifferenz $(*)$ direkt zu berechnen, indem man in (3.7) zunächst die Koeffizienten des Subtrahen-

den bestimmt und anschließend die Differenz der entsprechenden Polynomkoeffizienten bildet. Schließt man nämlich die Koeffizienten von Minuend und Subtrahend mit einer Langzahl-Intervallarithmetik ein, so erhält man quasi punktförmige Intervalle, die so weit getrennt liegen, daß ihre Differenzen nahezu ohne Überschätzungen berechnet werden können! Damit ist die Auswertung von $(*)$ auf die Auswertung nur eines Polynoms zurückgeführt, welche z.B. nach dem Intervall-Hornerschema geschehen kann, wenn man $|x - x_0| \leq \eta$ im Bedarfsfall zur Vermeidung von Überschätzungen beim Hornerschema noch in mehrere Teilintervalle unterteilt. Die Abschätzung des Restgliedes in der Lagrangeschen Form erfolgt durch automatische Differentiation und Auswertung der $(K+1)$-ten Ableitung von $H(x)$ über dem Intervall $|x - x_0| \leq \eta$, wobei eine Intervallunterteilung ebenfalls notwendig werden kann.

Es sei betont, daß das beschriebene Verfahren sehr deutlich zeigt, daß die naive Anwendung der Intervallrechnung nicht zum gewünschten Ziel führt, während beim gezielten Einsatz an der richtigen Stelle (Differenz der Polynom-Koeffizienten) die Intervallarithmetik ein äußerst nützliches Werkzeug ist!

3.2.2 Rationale Approximation

In diesem Abschnitt betrachten wir die Approximation einer vorgegebenen Hilfsfunktion $H(x)$ durch eine rationale Funktion und zeigen, wie eine garantierte Oberschranke des absoluten bzw. relativen Approximationsfehlers berechnet werden kann. Mit Hilfe eines XSC-Programms lassen sich diese Schranken automatisch berechnen.

Da das Lösungsverfahren die automatische Differentiation benötigt, wird vorausgesetzt, daß die Hilfsfunktion $H(x)$ als ein endlicher Ausdruck in den Funktionen

$$\text{exp, ln, sqr, sqrt, sin, cos, arctan, pow}$$

sowie den Grundoperationen $-(\text{unär}), +, -, *, /$ gegeben ist. (Solche Ausdrücke können derzeit im Modul `mpitaylor` bearbeitet werden.)

Die rationale Funktion zur Approximation von $H(x)$ möge

$$A_0 + A_1 \cdot (x - x_0)^1 + \ \ldots \ + A_N \cdot (x - x_0)^N$$

als Zählerpolynom und

$$B_0 + B_1 \cdot (x - x_0)^1 + \ \ldots \ + B_M \cdot (x - x_0)^M$$

als Nennerpolynom besitzen. Bei festen Polynomgraden N, M erhält man häufig eine sehr effektive Approximation, wenn die Koeffizienten A_j, B_j nach Tschebyscheff bestimmt werden. Der absolute oder relative Approximationsfehler besitzt dann im Innern des Approximationsintervalls $|x - x_0| \leq \eta$ mindestens $N + M$ relative Extremstellen mit oszillierenden aber betragsgleichen Extremwerten. Die Betragsgleichheit der Extremwerte kann jedoch in der Praxis aus folgenden Gründen nicht realisiert werden:

- Die A_j, B_j können nur mit endlich vielen Dezimalstellen berechnet werden.

- Es ist i. a. sinnvoll, die A_j, B_j zur nächstgelegenen Zahl in dem Raster zu runden, in dem die Polynome ausgewertet werden sollen.

Durch die notwendige Rundung der Polynomkoeffizienten werden die absoluten Extremwerte des Approximationsfehlers verschieden sein, und man wird i.a. auch nicht garantieren können, daß sich die Anzahl der Extremstellen nicht ändert.

Bezeichnet man die aus den A_j, B_j durch geeignete Rundung hervorgegangenen Koeffizienten mit a_j, b_j , so lauten die tatsächlich für die Approximation auf der Maschine verwendeten Polynome

$$\begin{aligned} \mathrm{P}_N(x - x_0) &:= a_0 + a_1 \cdot (x - x_0)^1 + \ \ldots \ + a_N \cdot (x - x_0)^N; \\ \mathrm{Q}_M(x - x_0) &:= b_0 + b_1 \cdot (x - x_0)^1 + \ \ldots \ + b_M \cdot (x - x_0)^M; \end{aligned}$$

d. h. $H(x)$ wird durch

$$H(x) \approx \frac{\mathrm{P}_N(x - x_0)}{\mathrm{Q}_M(x - x_0)}, \quad \mathrm{Q}_M(x - x_0) \neq 0, \quad |x - x_0| \leq \eta$$

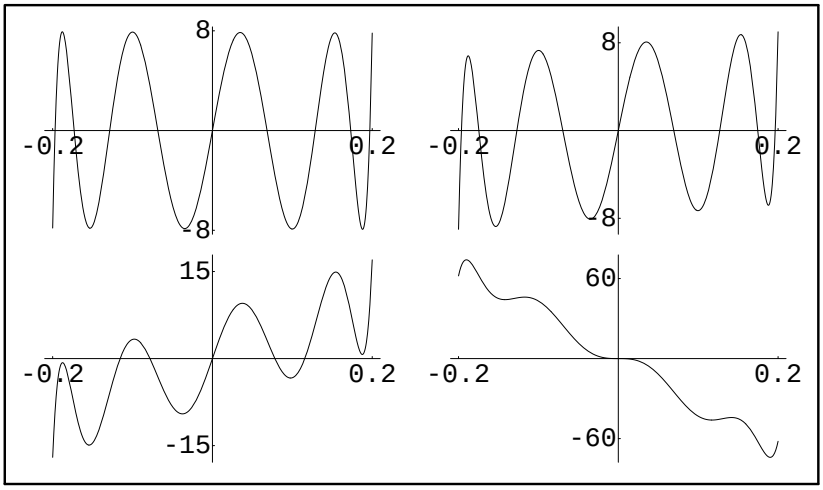
approximiert.

Das folgende Beispiel zeigt an Hand von vier Graphen den Einfluß der Rundung der Polynomkoeffizienten auf den Funktionsverlauf des relativen Approximationsfehlers.

Es wird $H(x) := e^x, \quad x_0 = 0, \quad \eta = 0.2, \quad M = N = 4;$ betrachtet. Die mit Mathematica berechneten Koeffizienten A_j, B_j lauten mit den ersten 18 Dezimalstellen

j	A_j	B_j
0	$9.99999999999999999\ldots \cdot 10^{-1}$	$+1.00000000000000000\ldots \cdot 10^{+0}$
1	$4.99999999999998228\ldots \cdot 10^{-1}$	$-4.99999999999998228\ldots \cdot 10^{-1}$
2	$1.07140305127468128\ldots \cdot 10^{-1}$	$+1.07140305127468128\ldots \cdot 10^{-1}$
3	$1.19034858976579290\ldots \cdot 10^{-2}$	$-1.19034858976579290\ldots \cdot 10^{-2}$
4	$5.95025533669726278\ldots \cdot 10^{-4}$	$+5.95025533669726278\ldots \cdot 10^{-4}$

Tabelle 3.1: Polynomkoeffizienten $A[j]$, $B[j]$ mit den ersten 18 Dezimalstellen



Die vier Graphen aus obiger Abbildung zeigen den mit 10^{+17} multiplizierten relativen Approximationsfehler, wenn die mit Mathematica berechneten Koeffizienten A_j, B_j auf 17, 16, 15 bzw.14 Dezimalstellen der a_j, b_j gerundet werden. Dabei erkennt man z.B., daß sich das Betragsmaximum des relativen Approximationsfehlers fast verachtfacht, wenn die Koeffizienten auf nur 14 Dezimalstellen gerundet werden.

Durch die in der Praxis notwendige Rundung der z.B. mit einer Langzahlarithmetik bestimmten Polynomkoeffizienten A_j, B_j sind die relativen Extremwerte nicht mehr betragsgleich, so daß eine auf die a_j, b_j bezogene Abschätzung des absoluten oder relativen Approximationsfehlers stets erforderlich ist.

3.2.3 Abschätzung des Approximationsfehlers

Nach Vorstellung des grundsätzlichen Lösungswegs im ersten Abschnitt und nach Demonstration des Rundungseinflusses der Polynomkoeffizienten auf den Approximationsfehler werden jetzt Formeln zusammengestellt, mit deren Hilfe garantierte Oberschranken des absoluten bzw. relativen Approximationsfehlers bzgl. der Näherung

$$H(x) \approx \frac{\mathrm{P}_N(x-x_0)}{\mathrm{Q}_M(x-x_0)}, \quad \mathrm{Q}_M(x-x_0) \neq 0, \quad |x-x_0| \leq \eta \tag{3.8}$$

(mit einem geeigneten XSC-Programm) berechnen werden können. Mit den Approximationspolynomen (die Koeffizienten sind Maschinenzahlen)

$$\mathrm{P}_N(x-x_0) := \sum_{j=0}^{N} a_j \cdot (x-x_0)^j, \quad \mathrm{Q}_M(x-x_0) := \sum_{j=0}^{M} b_j \cdot (x-x_0)^j \tag{3.9}$$

und der Taylorentwicklung der Hilfsfunktion $H(x)$ um den Punkt x_0

$$H(x) = \mathrm{T}_K(x-x_0) + \mathrm{R}_K(x) \tag{3.10}$$

$$\mathrm{T}_K(x-x_0) := \sum_{j=0}^{K} s_j \cdot (x-x_0)^j$$

$$\mathrm{R}_K(x) := \frac{H^{(K+1)}(\zeta)}{(K+1)!} \cdot (x-x_0)^{K+1}, \quad \zeta = \zeta(x) \text{ zwischen } x \text{ und } x_0 \tag{3.11}$$

gelten für den absoluten und relativen Approximationsfehler die Abschätzungen

$$|\Delta(x)| \leq \left|\frac{\mathrm{Q}_M(x-x_0)\cdot \mathrm{T}_K(x-x_0)-\mathrm{P}_N(x-x_0)}{\mathrm{Q}_M(x-x_0)}\right| + |\mathrm{R}_K(x)|\,,$$

$$|\varepsilon(x)| \leq \left|\frac{\mathrm{Q}_M(x-x_0)\cdot \mathrm{T}_K(x-x_0)-\mathrm{P}_N(x-x_0)}{H(x)\cdot \mathrm{Q}_M(x-x_0)}\right| + \left|\frac{\mathrm{R}_K(x)}{H(x)}\right|,$$
$$H(x)\neq 0\,.$$

Da sich in der Praxis die Polynomgrade N, M in der Regel höchstens um 1 unterscheiden werden, ist es keine wirkliche Einschränkung, wenn im folgenden $M+K \geq N$ vorausgesetzt sein soll. Die Grundidee ist nun, obiges Zählerpolynom zu berechnen, indem man die Differenzen z_j der Polynomkoeffizienten mit einer Intervall-Langzahlarithmetik auswertet

$$\mathrm{Q}_M\cdot \mathrm{T}_K - \mathrm{P}_N \equiv \mathrm{Z}_{M+K}(x-x_0) := \sum_{j=0}^{M+K} z_j\cdot (x-x_0)^j\,.$$

Durch Einsetzen erhält man dann die folgenden Abschätzungen:

$$|\Delta(x)| \leq \left|\frac{\mathrm{Z}_{M+K}(x-x_0)}{\mathrm{Q}_M(x-x_0)}\right| + |\mathrm{R}_K(x)|, \quad |x-x_0|\leq \eta, \tag{3.12}$$

$$|\varepsilon(x)| \leq \left|\frac{\mathrm{Z}_{M+K}(x-x_0)}{H(x)\cdot \mathrm{Q}_M(x-x_0)}\right| + \left|\frac{\mathrm{R}_K(x)}{H(x)}\right|, \quad H(x)\neq 0 \tag{3.13}$$

Zur Berechnung der Oberschranken von $|\Delta(x)|, |\varepsilon(x)|$ bzgl. $|x-x_0|\leq\eta$ sind die rechten Seiten von (3.12),(3.13) intervallmäßig auszuwerten. Wegen der dabei auftretenden Überschätzungen muß der Bereich $|x-x_0|\leq\eta$ in eine jeweils hinreichend große Anzahl von Intervallen unterteilt werden.

Abschätzung des Restgliedes in (3.12) bzw. in (3.13)
Nach (3.11) gelten mit

$$u := \frac{H^{(K+1)}([x_0-\eta, x_0+\eta])}{(K+1)!}$$

die Aussagen

$$\mathrm{R}_K(x)\in u\cdot[-\eta^{K+1}, +\eta^{K+1}],\ |\mathrm{R}_K(x)|\leq |u|\cdot\eta^{K+1} =: \tau,\ \mathrm{R}_K(x)\in[-\tau,+\tau].$$

Das Intervall u wird durch automatische Differentiation mit Hilfe des Moduls `mpitaylor` berechnet, wobei $[x_0-\eta, x_0+\eta]$ zur Vermeidung von Überschätzungen in hinreichend viele Teilintervalle $[x]_j$ unterteilt werden kann; in diesem Fall ist dann $|u|$ das Maximum der entsprechenden Teilintervall-Obergrenzen $|u_j|$.

In (3.13) muß $\mathrm{R}_K(x)$ noch durch $H(x)$ dividiert werden. Da bei der automatischen Differentiation die Funktionswerteinschließung $H(u_j)$ in jedem einzelnen Teilintervalle mitgeliefert wird, dividiert man dazu $|u|\cdot\eta^{K+1}$ noch durch das Minimum der Werte der berechneten $\langle\ |H([x]_j)|\ \rangle$. Falls dieses Minimum verschwindet, wird der Quotient auf `MaxReal` gesetzt, um anzuzeigen, daß eine Oberschranke des relativen Approximationsfehlers nicht berechnet werden kann.

Abschätzung des ersten Summanden in (3.12) bzw. in (3.13)

In (3.12) gelten zunächst mit der Abkürzung

$$v := \frac{\mathrm{Z}_{M+K}([x_0-\eta, x_0+\eta]-x_0)}{\mathrm{Q}_M([x_0-\eta, x_0+\eta]-x_0)}$$

die Abschätzung

$$\left|\frac{\mathrm{Z}_{M+K}(x-x_0)}{\mathrm{Q}_M(x-x_0)}\right| \le |v|.$$

Falls $|v|$ bei einem zu breiten Intervall $[x_0-\eta, x_0+\eta]$ wegen Überschätzungen bei den Polynomauswertungen von $\mathrm{Z}_{M+K}, \mathrm{Q}_M$ nach dem Intervall-Hornerschema zu groß ausfällt, muß $[x_0-\eta, x_0+\eta]$ wieder in eine hinreichend große Anzahl von Teilintervallen unterteilt werden.

In (3.13) muß im ersten Summanden rechts zusätzlich noch durch $H(x)$ dividiert werden, was bei komplizierteren Zielfunktionen die Laufzeit erheblich vergrößert. Dies kann man jedoch dadurch vermeiden, daß man $H(x)$ nach (3.10) durch ihre schon berechnete Taylordarstellung mit Restglied ersetzt:

$$|\varepsilon(x)| \le \left|\frac{\mathrm{Z}_{M+K}(x-x_0)}{[\mathrm{T}_K(x-x_0)+\mathrm{R}_K(x)]\cdot\mathrm{Q}_M(x-x_0)}\right| + \left|\frac{\mathrm{R}_K(x)}{H(x)}\right|, \quad H(x)\neq 0.$$

Da $|\mathrm{R}_K(x)|$ bereits durch τ abgeschätzt wurde, gilt $\mathrm{R}_K(x)\in[-\tau,+\tau]$, und der erste Summand rechts kann, wie im vorhergehenden Absatz beschrieben, für $x\in[x_0-\eta, x_0+\eta]$ intervallmäßig ausgewertet werden.

3.3 Anwendungsbeispiel

Als Anwendungsbeispiel wird die bereits im Jahre 1968 von Hart et. al. [43] mit der Bezeichnung `JZERO 5847` vorgeschlagene rationale Approximation untersucht. Diese ist laut Angabe der Autoren auf eine Genauigkeit[1] von 16,59 Dezimalstellen ausgelegt. Es handelt sich hierbei um eine Approximation

$$J_0(x) \approx \frac{P(x^2)}{Q(x^2)}$$

im Approximationsintervall $[0, 8]$. Dabei ist P ein Polynom vom Grade 10 mit den Koeffizienten

```
p[0] :=  0.58828674632868341423835960 9e+11
p[1] := -0.14343701164751473694917554 98e+11
p[2] :=  0.82932752137525404768994811 64e+9
p[3] := -0.20101723394107941321672047 49e+8
p[4] :=  0.25640543425244074556679422 13e+6
p[5] := -0.19344679360067813003600036 78e+4
p[6] :=  0.91931791066585313639377354 89e+1
p[7] := -0.28251741814863245632750079 24e-1
p[8] :=  0.55529137316071119492993991 24e-4
p[9] := -0.64943670893011936569757019 41e-7
p[10]:=  0.35307722170456043917811230 86e-10
```

und Q ein Polynom vom Grade 4 mit den Koeffizienten

```
q[0]:= 0.58828674632868342934662993 76e+11
q[1]:= 0.36346749346560087410642370 87e+9
q[2]:= 0.99635360310006026750272778 24e+6
q[3]:= 0.14643417762555995396789435 142e+4
q[4]:= 0.1e+1
```

Für die Analyse des Approximationsfehlers werden die angegebenen dezimalen Polynomkoeffizienten jeweils zunächst zur nächstgelegenen binären Gleitkommazahl des IEEE double–Zahlformats gerundet. Man erhält nun die folgenden Koeffizienten (Angabe jeweils als exakte hexadezimale Zahl und als gerundete Dezimalzahl):

```
Polynomkoeffizienten des Zaehlerpolynoms:
422B64ECAC91BC97    5.882867463286834E+010
```

[1] engl. Angabe: Precision

```
C20AB79C15660305   -1.434370116475147E+010
41C8B74450B00853    8.293275213752540E+008
C1732BA5B64E4421   -2.010172339410794E+007
410F4CAB795957FA    2.564054342524408E+005
C09E39DF2A9DD6FC   -1.934467936006781E+003
402262E85F32BCDF    9.193179106658532E+000
BF9CEE064C997865   -2.825174181486325E-002
3F0D1CFEA1466E77    5.552913731607112E-005
BE716EE52408A9C8   -6.494367089301193E-008
3DC3691EC15A1E18    3.530772217045604E-011

Polynomkoeffizienten des Nennerpolynoms:
422B64ECAC91BC97    5.882867463286834E+010
41B5AA12E577319E    3.634674934656008E+008
412E680334C987F6    9.963536031000603E+005
4096E15DFA984166    1.464341776255600E+003
3FF0000000000000    1.000000000000000E+000
```

Zur einfacheren programmtechnischen Umsetzung wird anstelle der Approximation

$$J_0(x) \approx \frac{P(x^2)}{Q(x^2)}, \quad x \in [0, 8]$$

die äquivalente Approximation

$$J_0(\sqrt{y}) \approx \frac{P(y)}{Q(y)}, \quad y \in [0, 64]$$

verwendet.

Das vollständige Programm ist im Anhang B.2 angegeben.

Die Abbildung 3.1 zeigt den Fehlerverlauf der Approximation `JZERO 5847`. Man beachte, daß die Funktionswerte mit `PlotScale` $= 10^{16}$ skaliert sind. Als Schranken für die maximalen Approximationsfehler (Verwendung von 10 Millionen Teilintervallen) findet man den Wert

```
1.349859480357124E-013.
```

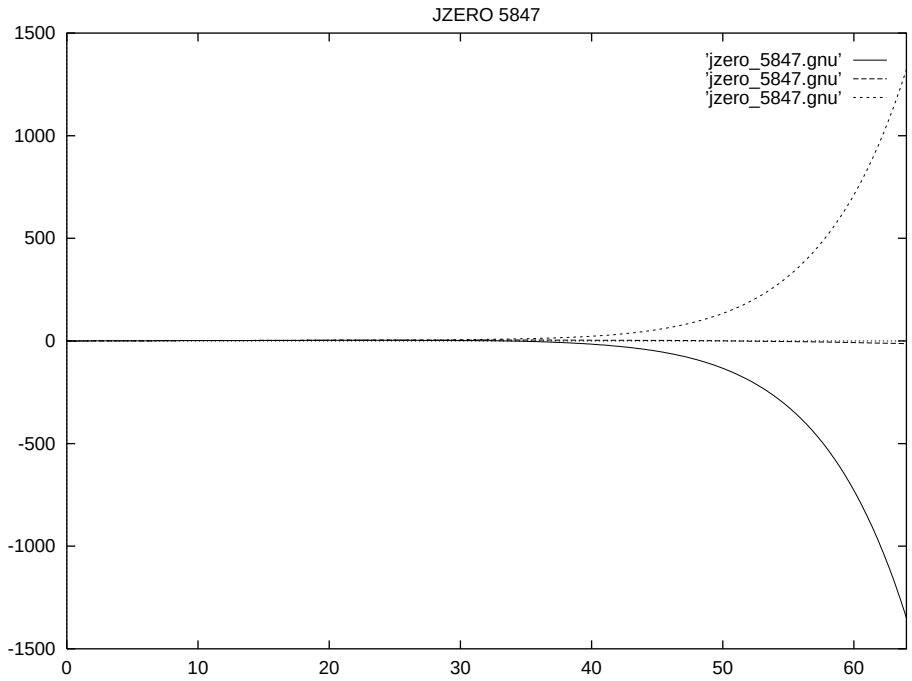


Abbildung 3.1: Fehlerkurve für JZERO 5847

Die mittlere Kurve in Abbildung 3.1 ergeben sich, wenn man zusätzlich zu den Begrenzungsbildpunkten noch einen Funktionswert der Fehlerkurve an einer zufällig gewählten Stelle in jedem einer Spalte entsprechenden Abszissenintervall einzeichnet.

Die Begrenzungsfunktionen, die die Fehlerkurve einschließen, sind Treppenfunktionen. Dies wird im symbolischen Schaubild 3.2 verdeutlicht.

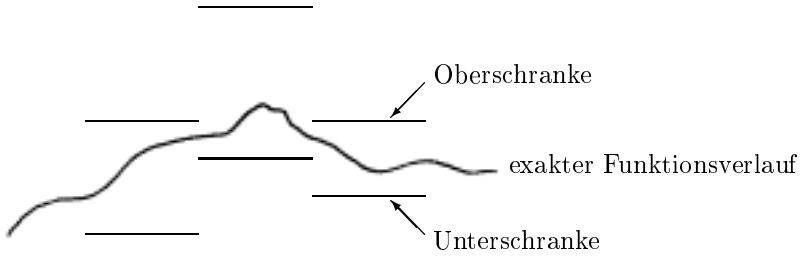


Abbildung 3.2: Einschluß einer Fehlerkurve

Dem folgenden Auszug aus dem Ausgabeprotokoll des Programms kann die oben angegebene maximale absolute Fehlerschranke für den Approximationsfehler entnommen werden:

```
Approximationsintervall =  0.000000000000000E+000   6.400000000000000E+001
Funktionsauswertung ueber Gesamtintervall:
[              -6.5E+001,                5.0E+001 ]
Entwicklungsstelle x0 =  0.000000000000000E+000
Mini: -1.000000000000000E-022   Maxi:  1.000000000000000E-022
Mini: -1.371089907057216E-015   Maxi:  1.000000000000000E-022
Gewaehlte Anzahl von Teilintervallen: 10000000
Breite eines Teilintervalls:  6.400000000000000E-006

Endergebnis:
Mini:   -1.349859480357124E-013      Maxi:  1.322437650989959E-013
MiniUb:  1.322437650989959E-013    MaxiLb: -1.349859480357124E-013
```

Eine andere rationale Approximation für $J_0(x)$ läßt sich mit Hilfe des Computeralgebrasystems Maple finden. Bei Verwendung der Maple-Funktion `minimax`[2] mit den gleichen Vorgaben (Approximationsbereich $[0, 64]$, rationale Approximation, Grad des Zählerpolynoms $= 10$, Grad des Nennerpolynoms $= 4$) ergeben sich die folgenden, in das IEEE double–Format gerundeten Koeffizienten:

```
Polynomkoeffizienten des Zaehlerpolynoms:
3FEA1AC2D1432CCB    8.157667243152898E-001
BFC9759B11994A04   -1.989015422755785E-001
```

[2] Approximation mittels Remez-Algorithmus

```
3F878D623598C74E    1.150013662399698E-002
BF32449948DAA553   -2.787470083186100E-004
3ECDD370642CA386    3.555528367189497E-006
BE5CCD94089B5721   -2.682492142895833E-008
3DE1854F52299C23    1.274801731895777E-010
BD5B915AB121AE7D   -3.917618599143659E-013
3CCBBE1CEE0C8CC9    7.700126351795405E-016
BC309CC902971681   -9.005622946037904E-019
3B827F2D74BA4BC3    4.896058845739470E-022

Polynomkoeffizienten des Nennerpolynoms:
3FEA1AC2D1432CCB    8.157667243152898E-001
3F74A4F7F53C581F    5.040138803243748E-003
3EECF98BF497CF2C    1.381625738170569E-005
3E55CD9B3AB5BCC9    2.030576566391450E-008
3DAE7E538A39AB09    1.386682116315572E-011
```

Als absolute Fehlerschranke erhält man einen Wert in gleicher Grössenordnung:

```
1.379131175072762E-013.
```

Ein weiteres Anwendungsbeispiel ist im Abschnitt 5.6 dargestellt.

Kapitel 4

Spezielle Funktionen

In diesem Kapitel wird ein kurzer Überblick über bereits existierenden Softwareimplementierungen von speziellen Funktionen der mathematischen Physik gegeben (siehe auch Hofschuster/Krämer [53]). Als „spezielle Funktionen“ werden in dieser Arbeit Funktionen[1] wie z.B. $\Gamma(x)$, Polygamma–, Fehler–, Bessel–Funktionen, Dawsonsches Integral usw. bezeichnet, in Abgrenzung zu den sogenannten „elementaren Funktionen“[2] wie z.B. $\sqrt{x}$, exp, expm1, log, log1p, exp2, exp10, log2, log10, sin, cos, tan cot, arcsin, arccos, arctan, arccot, sinh, cosh, tanh, coth, arsinh, arcosh, artanh, arcoth.

4.1 Vorhandene Implementierungen

Die vorhandene Software kann zunächst grundsätzlich in kommerzielle und nichtkommerzielle Programmprodukte eingeteilt werden. Zu den kommerziellen Produkten können sowohl große bekannte Funktionsbibliotheken und Computeralgebrasysteme als auch Buchveröffentlichungen mit beigelegter Software gezählt werden. Bei den nichtkommerziellen Produkten handelt es sich meist um Softwarebibliotheken, die frei im Internet erhältlich sind.

4.1.1 Kommerzielle Bibliotheken

Einige wichtige kommerzielle Softwareprodukte, in denen auch Implementierungen von speziellen Funktionen der mathematischen Physik enthalten

[1] Eine Auflistung einiger spezieller Funktionen befindet sich im Anhang C.

[2] In der deutschsprachigen Literatur teilweise auch als „Standardfunktionen“ bezeichnet.

sind, werden im folgenden aufgeführt:

1. (Funktions-)Bibliotheken:

- NAG Library (für Fortran 77, Fortran 90, C)
- IMSL MATH Library (für Fortran 77, Fortran 90, C)
- CERN Library (für Fortran 77)
- Mathematical Function Library für Mircosoft Fortran oder C, (United Laboratories)

2. Computeralgebrasysteme und Programmpakete:

- Matlab
- Maple
- Mathematica
- Macsyma
- Derive
- Mupad
- Reduce

3. Buchveröffentlichungen mit beigelegter Software:

- Press et. al.: Numerical Recipes, (Basic, C, Fortran, Pascal), 1989 [99]
- Thompson: Atlas for Computing Mathematical Functions (Fortran90, Mathematica), 1997 [117]
- Zhang/Jin: Computation of Special Functions (Fortran), 1996 [124]

4.1.2 Nichtkommerzielle Bibliotheken

Eine Vielzahl von Softwarepaketen kann im Internet gefunden werden. Sie liegt üblicherweise im Quellcode vor und darf je nach Lizenz („public domain“, „gnu public licence“ der „Free Software Foundation“) in der Regel frei benutzt werden. Wichtige Programmpakete im Zusammenhang mit speziellen Funktionen sind:

- AMOS (Amos, D.E. - Sandia National Laboraties), Fortran
- Cephes (Moshier, S.L. - Cambridge), C Bibliothek
  6 verschiedene Zahlenformate mit unterschiedlichen Genauigkeiten
- FUNLIB/FNLIB (Fullerton, W.), Fortran
- FUNPACK (Cody, W. J.), Fortran;
  Exponentielles, elliptisches, Dawsonsches Integral, Besselfunktionen

- SPECFUN (Cody, W. J.), Fortran Bibliothek für spezielle Funktionen; Gamma–, Fehler–, Besselfunktionen
- SLATEC (Fullerton, W.)
- Mathematical Software Package (Ooura - Univ. Tokyo)
- MISCFUN (Macleod, A.J. - U. Paisley),
  Fortran Bibliothek für seltenere spezielle Funktionen;
  Integrale von Bessel– und Airy–Funktionen, Struve–Funktionen, ...
- VFNLIB (Boisvert, R., Saunders, B.),
  Vectorized evaluation of special functions
- Transactions of Mathematical Software (TOMS),
  viele Routinen für spezielle Funktionen

Weiter enthalten einige mathematische Bibliotheken von C–Compilern spezielle Funktionen (in der Regel Gamma–, Fehler– und Besselfunktionen). Der Quelltext der folgenden beiden Bibliotheken ist z.B. frei zugänglich:

- fdlibm (SunSoft, Dr. Ng),
  „freely distributable standard math library“
- Mathematische Bibliothek „libm“ des GNU C–Compilers

Gute Startpunkte für die Suche nach Programmcode für spezielle Funktionen im Internet sind z.B.

- NETLIB: Große umfangreiche Sammlung mathematischer Software und verschiedenen Veröffentlichungen
  `http://www.netlib.org` und `ftp://ftp.netlib.org`
- GAMS (Guide to available mathematical software): Verzeichnis mathematischer Software, `http://gams.nist.gov`
- ELIB (Electronic Library for Mathematical Software): Sammlung mathematischer Software, `http://elib.zib.de`
- TOMS: ACM Transactions on Mathematical Software, (u.a. in NETLIB integriert)

Insgesamt kann man heute eine ganze Reihe von Softwareimplementierungen für elementare und spezielle Funktionen im Internet finden, viele sind sogar im Quelltext verfügbar.

4.1.3 Anwendbarkeit in der Verifikationsnumerik

Im Zusammenhang mit der Verifikationsnumerik und der Intervallrechnung stellt sich die Frage, wie zuverlässig die verfügbaren Softwareimplementierungen sind und ob im mathematischen Sinne bewiesene Fehlerschranken

bekannt sind. Mit anderen Worten: Können diese Implementierungen und Bibliotheksfunktionen ohne große Änderungen im Bereich der Numerik mit automatischer Ergebnisverifikation eingesetzt werden?

Eine genauere Untersuchung zeigt, daß sich bisher weder im Quellcode der Bibliotheksfunktionen noch in der zugehörigen Dokumentation (soweit diese existiert) Aussagen über relative Fehlerschranken finden. Der einzige Anhaltspunkt sind Fehlerschätzungen, die von manchen Programmautoren angegeben werden.

Exemplarische soll hier ein Auszug aus dem sogenannten „readme-file“ (Begleittext mit wichtigen Hinweisen) des qualitativ hochwertigen Softwarepacketes `fdlibm` von SunSoft gegeben werden (in englischer Sprache):

```
FDLIBM is intended to provide a reasonably portable (see assumptions
below), reference quality (below one ulp for major functions like
sin,cos,exp,log) math library (libm.a).
```

No error **bounds** are given for the special functions `erf`, `gamma`, `j0`, `j1` and `jn`. In this library other special functions are not available.

Als weiteres Beispiel für das Fehlern von rigorosen (relativen) Fehlerschranken können die folgenden Kommentarzeilen der bekannten und umfangreichen `cephes`–Bibliothek dienen (ebenfalls in englischer Sprache):

```
 * Bessel function of noninteger order    y = jv( v, x );
 * ACCURACY:
 * Error criterion is absolute, except relative when |jv()| > 1.
 * arithmetic  v domain  x domain    # trials      peak       rms
 *    IEEE      0,125     0,125      100000      4.6e-15    2.2e-16
 *    IEEE   -125,0       0,125       40000      5.4e-11    3.7e-13
 *    IEEE      0,500     0,500       20000      4.4e-15    4.0e-16
```

Es werden lediglich für eine bestimmte Anzahl (`trials`) von Zufallszahlen Funktionsauswertungen durchgeführt. Die berechneten Funktionsergebnisse werden mit den Ergebnissen einer (in der Regel höhergenauen) Referenzfunktion verglichen, der dabei maximal beobachtete Fehler (`peak`) und der durchschnittlich auftretende Fehler (`rms`) werden angegeben. Wiederrum werden aber keine Fehlerschranken genannt.

Die in den beiden Beispielen gezeigten Angaben sind auch für die anderen Funktionsbibliotheken typisch, die im Zusammenhang mit dieser Arbeit untersucht wurden. Aus den mitgelieferten Angaben ist es nicht möglich auf einfache Art und Weise rigorose Fehlerschranken herzuleiten.

Es zeigt sich, daß zur Zeit eine Vielzahl an guten Softwarepaketen zur Berechnung von speziellen Funktionen der mathematischen Physik erhältlich sind. Dennoch können diese Softwarepakete nicht im Zusammenhang mit Verifikationsnumerik eingesetzt werden, da keine rigorosen mathematisch bewiesenen Fehlerschranken bekannt sind, die jeweils für alle zulässigen Gleitkommaargumente gelten.

4.2 Funktionsroutinen mit sicheren Fehlerschranken

Bei der Entwicklung und Implementierung einer Funktion mit sicheren Fehlerschranken kann nach folgenden Schritten vorgegangen werden:

1. Erstellung einer nahezu perfekten Intervallapproximation der zu implementierenden Funktion. Die hierzu benötigte Fehlerschranke muß in der Regel analytisch gefunden werden.
2. Berechnung von Einschließungen der Nullstellen der Funktion.
3. Herleitung von Formeln zur Argumentreduktion (falls möglich).
4. Bestimmung geeigneter (Best–)Approximationen, z.B. mit Hilfe des Remez–Algorithmus, die Nullstellen der Funktion (Schritt 2) sind geeignet zu berücksichtigen.
5. Bestimmung des Approximationsfehlers, hierzu kann die nahezu perfekte Approximation (aus Schritt 1) zusammen mit entsprechenden Intervall–Softwarewerkzeugen verwendet werden (siehe auch Kapitel 3).
6. Herleitung von Formeln zur Ergebnisanpassung (falls nötig, siehe Schritt 3).
7. Bestimmung der Fehlerschranke („worst case error bound“) für den kompletten Algorithmus. Hierbei sind alle auftretenden Fehlerarten zu berücksichtigen, d.h. Rundungsfehler, Approximationsfehler, Fehler in den Eingangsdaten.

Für den Fall einer elementaren Funktion sind die Schritte 1 und 2 im allgemeinen sehr einfach, für die Schritte 3 und 6 werden bekannte Formeln verwendet.

Der bisher verfügbare Programmcode zur Berechnung spezieller Funktionen enthält in der Regel nur die Ergebnisse der Schritte 3, 4 und 6. Im Rahmen der Verifikationsnumerik werden aber Fehlerschranken benötigt, Fehlerschätzungen, wie sie teilweise angegeben werden, genügen nicht. Die aufwendigsten und wichtigsten Schritte bei der Implementierung einer Funktion, die im Rahmen der Verifikationsnumerik eingesetzt werden soll, sind allerdings die Schritte 1, 5 und 7. Da eine Umsetzung der Schritte 5 und 7 per Hand meist sehr aufwendig und damit auch fehleranfällig sind, sollten hier geeignete Softwarewerkzeuge eingesetzt werden. Hierzu eignet sich der im Kapitel 2 vorgestellte Fehlerkalkül und die im Kapitel 3 aufgezeigte Methode zur Approximationsfehlerabschätzung. Die programmtechnische Umsetzung einer Bibliotheksfunktion ist nur ein kleiner Bruchteil der nötigen Arbeit.

Die Problematik der Umsetzung der Schritte 5 und 7 von Hand ohne geeignete Softwarewerkzeuge kann an der hervorragenden Arbeit von Tang [114] gesehen werden. Tang hat von Hand Fehlerschranken für seine Algorithmen zur Berechnung einiger elementarer Funktionen hergeleitet. Er gibt in seiner Arbeit z.B. für ein Tabellenverfahren zur Berechnung der Logarithmusfunktion eine Fehlerschranke von 0.56 ulp an. Eine Implementierung seines Algorithmus mit den von ihm angegebenen Konstanten führt zu den folgenden Rechenergebnissen, die mit einem wesentlich grösseren Fehler behaftet sind, als es durch die Fehlerschranke vorgegeben ist. Teilweise sind lediglich ein oder zwei Dezimalstellen des berechneten Ergebnisses richtig:

```
x:= 1.56640625              3FF9100000000000 (hex)
  ln(x) korrekt                 :  0.44878398...
  Ergebnis Algorithmus von Tang:  0.43804179...
x:= 3.1350000               4009147AE147AE14 (hex)
  ln(x) korrekt                 :  1.1426291...
  Ergebnis Algorithmus von Tang:  1.1318869...
```

Eine genauere Betrachtung zeigt, daß drei der 128 Tabellenwerte nur auf wenige Ziffern korrekt sind. Eine Korrektur dieser drei Werte kann Krämer [69] entnommen werden.

Es ist nötig, alle Fallunterscheidungen (im obigen Beispiel 128 verschiedene Bereiche) bei der Bestimmung der Fehlerschranke mitzubetrachten. Hierzu sollten Intervall–Softwarewerkzeuge eingesetzt werden. Zur Berech-

nung von numerischen Werten für benötigte Konstanten sollten Intervall–Langzahlarithmetiken (siehe Abschnitt 1.1) eingesetzt werden.

Im Bereich der speziellen Funktionen der mathematischen Physik gibt es bisher nur sehr wenig Publikationen, die Algorithmen bzw. Implementierungen mit garantierten Fehlerschranken für einzelne Funktionen angeben. Genannt seien hier die Veröffentlichungen von Barth/Krämer [13], Blomquist/Krämer [18], Krämer [60, 61], Luther/Otten [86] und Werner [122]. Insbesondere existiert keine einheitliche Softwarebibliothek derartiger Funktionen. Dies erklärt sich zum Teil aus den in diesem Bereich noch offenen und schwierigen Fragestellungen. Weiter sind diffizile Fehlerabschätzungen durchzuführen. Aufgrund der Komplexität dieser Fehlerabschätzungen lassen sich weite Teile hiervon praktisch nur auf dem Computer ausführen. Hierbei werden umfangreiche Softwarewerkzeuge benötigt (vgl. Kapitel 3 und 2).

Kapitel 5

Besselfunktionen

Im ersten Teil dieses Kapitels werden zunächst die Besselfunktionen[1] eingeführt und es werden wichtige Eigenschaften dieser Funktionen angegeben. Im zweiten Teil folgt einer kurzen Vorstellung einiger vorhandener Softwareimplementierungen eine ausführliche Untersuchung verschiedener Möglichkeiten zur Berechnung von Einschließungen von Funktionswerten der Besselfunktionen.

5.1 Mathematische Grundlagen

5.1.1 Besselsche Differentialgleichung

Die Besselsche Differentialgleichung[2] der Ordnung ν hat die Gestalt

$$z^2 \frac{d^2w}{dz^2} + z\frac{dw}{dz} + (z^2 - \nu^2)w = 0,$$

wobei ν ein komplexer Parameter mit $\mathrm{Re}\nu \geq 0$ ist. Es handelt sich hierbei um eine lineare Differentialgleichung zweiter Ordnung, die Stelle $z = 0$ ist schwach singulär und die Stelle $z = \infty$ ist stark singulär. Im folgenden wird die Besselsche Differentialgleichung im Körper $\mathbb{R}$ der rellen Zahlen

[1] Benannt nach Friedrich Wilhelm Bessel, 1784-1846, Königsberger Astronom. Er stieß 1824 auf die Besselsche Differentialgleichung, als er die Störungen der Planetenbahnen untersuchte, sie kommt allerdings schon rund hundert Jahre früher bei Daniel Bernoulli und Leonhard Euler im Zusammenhang mit Schwingungsproblemen vor.

[2] Das „Problem der schwingenden Membran“ führt auf die Besselsche Diffentialgleichung (siehe z.B. Heuser [44], Seite 292ff).

betrachtet, sie kann dann in der Form

$$x^2y'' + xy' + (x^2 - v^2)y = 0, \quad (v \in \mathbb{R})$$

geschrieben werden. Die Ordnungen $v := 0, 1/2$ und 1 sind hierbei wichtige Sonderfälle. Die Lösungen dieser Differentialgleichung werden als Besselsche Funktionen (kurz: Besselfunktionen) bezeichnet.

Zur Herleitung der Lösung dieser Differentialgleichung wird auf Walter [120] verwiesen. Eine gute einführende Darstellung der Besselschen Differentialgleichung und der Besselfunktionen findet sich in Heuser [44]. Ein umfassendes Werk stammt von Watson [121].

5.1.2 Besselsche Funktionen

Die Besselsche Funktion erster Art[3] der Ordnung v kann durch die Reihendarstellung

$$\begin{aligned} J_v(x) &:= \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{2n+v} n! \Gamma(n+v+1)} x^{2n+v} \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n! \Gamma(n+v+1)} \left(\frac{x}{2}\right)^{2n+v} \end{aligned}$$

definiert werden. Für $v \in \mathbb{N}_0$ ist J_v eine auf ganz $\mathbb{R}$ definierte reellwertige Funktion. Ist v (≥ 0) *keine* ganze Zahl, so ist

$$\begin{aligned} J_{-v}(x) &:= \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{2n-v} n! \Gamma(n-v+1)} x^{2n-v} \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n! \Gamma(n-v+1)} \left(\frac{x}{2}\right)^{2n-v} \end{aligned}$$

eine zweite, von $J_v(x)$ linear unabhängige Lösung der Besselschen Differentialgleichung.

[3] auch: Besselfunktionen erster Gattung

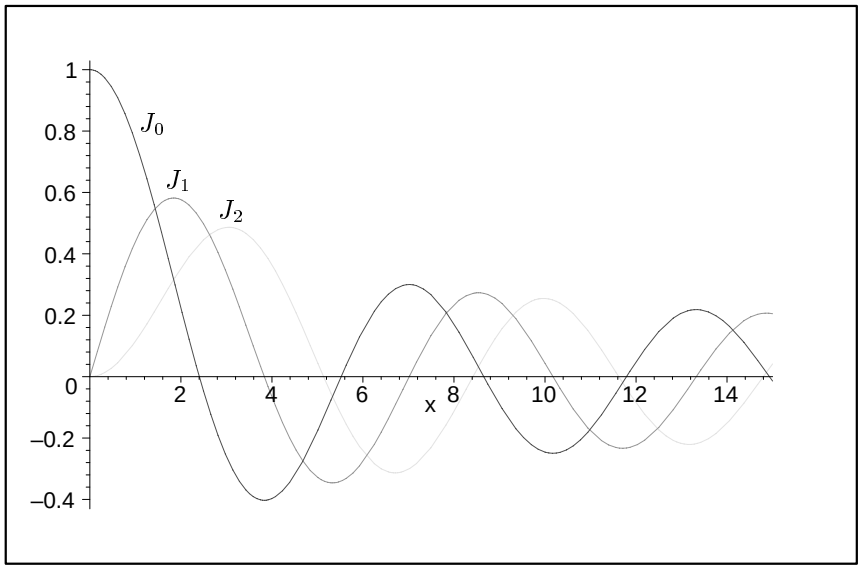


Abbildung 5.1: Besselfunktionen erster Art: J_0, J_1 und J_2

Satz 5.1.1 Ist der exponierte Index $v \geq 0$ der Besselschen Differentialgleichung keine ganze Zahl, so wird ihre allgemeine Lösung auf $x > 0$ gegeben durch

$$Z_v(x) = c_1 J_v(x) + c_2 J_{-v}(x), \quad c_1, c_2 \text{ bel. Konstanten.}$$

Beweis: Heuser [44, Beweis zu Satz 28.1]

Die Besselsche Funktion zweiter Art der Ordnung v kann mit Hilfe der Besselschen Funktion erster Art definiert werden.:

$$Y_v(x) := \frac{2}{\pi}\left(y_2(x) + (C - \ln 2) J_v(x)\right) \quad (x > 0, v \in \mathbb{N}_0)$$

mit $C := 0.57721\ldots$ (Euler-Mascheronische Konstante) und

$$\begin{aligned} y_2(x) \quad := \quad & J_v(x) \ln x - \frac{1}{2} \sum_{n=0}^{v-1} \frac{(v-n-1)!}{n!} \left(\frac{x}{2}\right)^{2n-v} \\ & -\frac{1}{2} \sum_{n=0}^{\infty} \frac{(-1)^n (h_n + h_{n+v})}{n!(n+v)!} \left(\frac{x}{2}\right)^{2n+v}, \quad (x > 0; v \in \mathbb{N}) \end{aligned}$$

sowie

$$h_n := 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

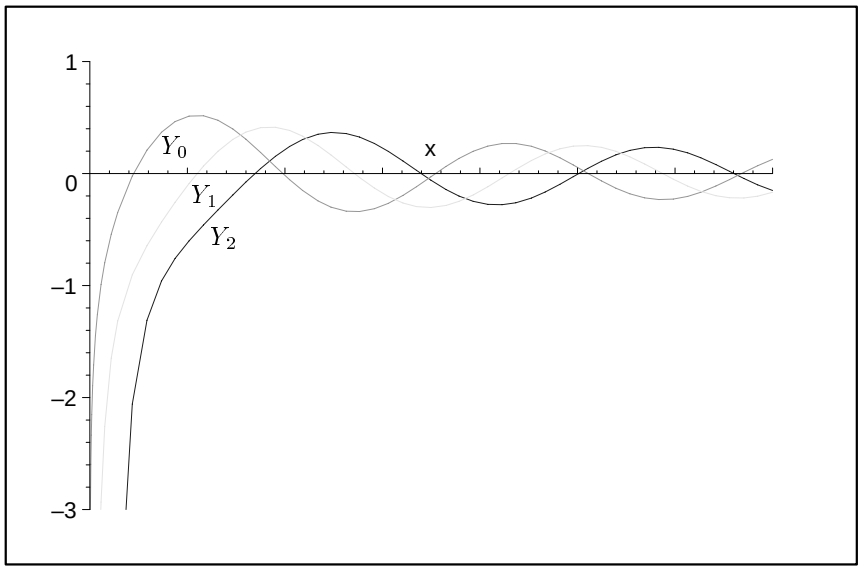


Abbildung 5.2: Besselfunktionen zweiter Art: Y_0, Y_1 und Y_2

Satz 5.1.2 Ist der exponierte Index $v \geq 0$ der Besselschen Differentialgleichung ganzzahlig, so wird ihre allgemeine Lösung auf $x > 0$ gegeben durch

$$Z_v(x) = c_1 J_v(x) + c_2 Y_v(x), \quad c_1, c_2 \text{ bel. Konstanten.}$$

Für $x \to 0$ ist $J_v(x)$ beschränkt, nicht jedoch $Y_v(x)$.
Definition:

$$J_{-v}(x) := (-1)^v J_v(x), \qquad \text{falls } v \in N$$

Anmerkung: Besselfunktionen als Lösungen der komplexen Differentialgleichung sind komplexwertige Funktionen. Die Abbildungen 5.3 und 5.4 sollen das Verhalten der Besselfunktionen über der komplexen Zahlenebene exemplarisch für J_0 veranschaulichen. Die Abbildungen 5.5 und 5.6 zeigen die Funktion Y_6.

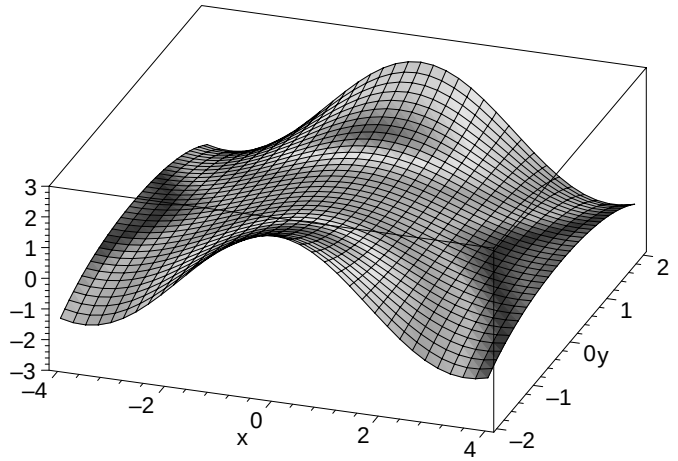


Abbildung 5.3: Realteil der Besselfunktion J_0

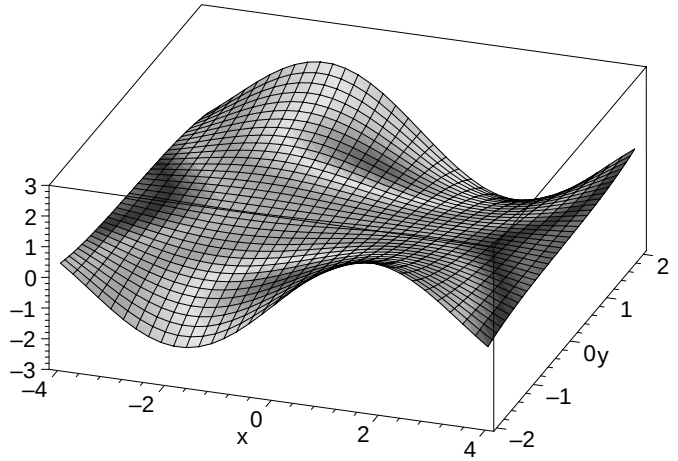


Abbildung 5.4: Imaginärteil der Besselfunktion J_0

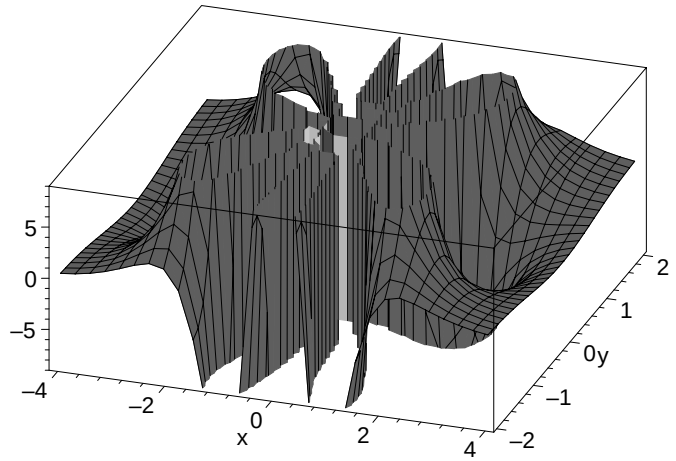


Abbildung 5.5: Realteil der Besselfunktion Y_6

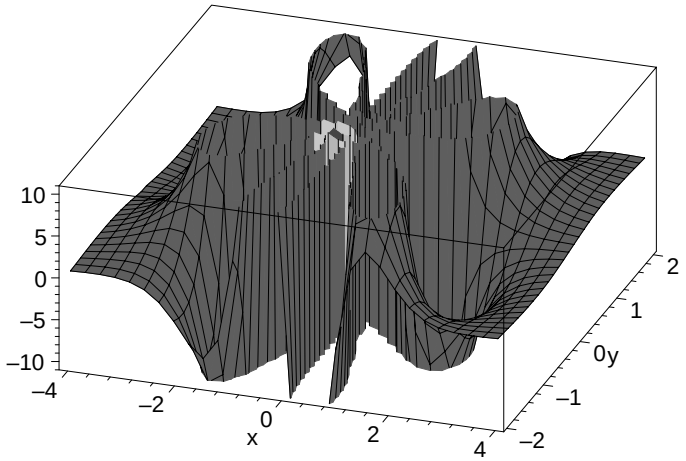


Abbildung 5.6: Imaginärteil der Besselfunktion Y_6

5.1.3 Eigenschaften der Besselschen Funktionen

Die folgenden Sätze 5.1.3 bis 5.1.6 beschreiben wichtige Eigenschaften der Besselschen Funktionen. Die entsprechenden Beweise finden sich bei Heuser [44].

Satz 5.1.3 *(Differentiationssatz)* Auf der Halbgeraden $x > 0$ ist

$$\frac{d}{dx}(x^v J_v(x)) = x^v J_{v-1}(x),$$

$$\frac{d}{dx}(x^{-v}J_v(x)) = -x^{-v}J_{v+1}(x),$$

insbesondere gilt $J_0'(x) = -J_1(x)$.

Satz 5.1.4 *(Rekursionssatz)* Für $x > 0$ ist

$$xJ_{v+1} - 2vJ_v + xJ_{v-1} = 0$$

und

$$2J_v' = J_{v-1} - J_{v+1}.$$

Satz 5.1.5 *(Integraldarstellung)* Für alle $n = 0, 1, 2, \ldots$ und alle x ist

$$J_n(x) = \int_0^\pi \cos(x\sin\phi - n\phi)d\phi.$$

Satz 5.1.6 *(Nullstellensatz)* Die Funktion J_n ($n \in \mathbb{N}_0$) hat abzählbar viele positive Nullstellen. Sie sind allesamt einfach und haben nur den einen Häufungspunkt ∞. Die Nullstellen > 0 von $J_0, J_1, \ldots$ trennen sich wechselseitig, schärfer: zwischen je zwei aufeinanderfolgenden Nullstellen von J_n liegt genau eine von J_{n-1} und genau eine von J_{n+1}.

Die Nullstellen der Besselschen Funktionen für $n \geq 0$ werden folgendermassen bezeichnet:

$j_{n,k}$ sei k-te positive Nullstelle von $J_n(x)$,

$y_{n,k}$ sei k-te positive Nullstelle von $Y_n(x)$.

Für die Nullstellen gilt mit diesen Bezeichnungen die Ungleichungskette:

$$n < y_{n,1} < j_{n,1} < y_{n,2} < \ldots < y_{n,k} < j_{n,k} < \ldots \ .$$

Wichtige Eigenschaften der Besselfunktionen
In den folgenden 8 Punkten werden einige wichtige Eigenschaften der Besselfunktionen zusammengefaßt (n ganzzahlig, $x \in \mathbb{R}$):

1. Wronski-Determinante

$$J_{n+1}(x)Y_n(x) - J_n(x)Y_{n+1}(x) = \frac{2}{\pi x}$$

2. Rekursion (3-gliedrig)

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

3. Ableitungen

$$\frac{d}{dx}(x^{-n} J_n(x)) = -x^{-n} J_{n+1}(x), \quad J_0'(x) = -J_1(x)$$

$$J_n'(x) = \frac{1}{2}(J_{n-1}(x) - J_{n+1}(x))$$

$$Y_n'(x) = \frac{1}{2}(Y_{n-1}(x) - Y_{n+1}(x))$$

4. Erzeugende Funktion

$$\cos(x \sin s) = J_0(x) + 2 \sum_{n=1}^{\infty} J_{2n}(x) \cos(2ns)$$

Spezialfall $s := 0$

$$1 = J_0(x) + 2 \sum_{n=1}^{\infty} J_{2n}(x)$$

5. Asymptotische Form ($x \gg v$)

$$J_v(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \frac{1}{2} v\pi - \frac{1}{4}\pi\right)$$

$$Y_v(x) \approx \sqrt{\frac{2}{\pi x}} \sin\left(x - \frac{1}{2} v\pi - \frac{1}{4}\pi\right)$$

6. Kettenbruch

$$\frac{J_n(x)}{J_{n-1}(x)} = \frac{x}{2n-} \frac{x^2}{2(n+1)-} \frac{x^2}{2(n+2)-} \cdots$$

7. Integraldarstellung

$$J_0(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin\theta)d\theta$$

$$Y_0(x) = \frac{1}{\pi}\int_0^\pi \sin(x\sin\theta)d\theta$$

8. Symmetrie

$$J_n(-x) = (-1)^n J_n(x)$$
$$J_{-n}(x) = (-1)^n J_n(x)$$
$$Y_{-n}(x) = (-1)^n Y_n(x)$$

Die Besselfunktionen können, auch im Hinblick auf eine spätere Implementierung, folgendermaßen in Gruppen eingeteilt werden:

1. Funktionen der Ordnung 0 oder 1 mit reellem Argument

   $J_0(x), J_1(x), Y_0(x), Y_1(x), I_0(x), I_1(x), K_0(x), K_1(x)$

2. Funktionen ganzzahliger Ordnung und reellem Argument

   $J_n(x), Y_n(x), I_n(x), K_n(x)$

3. Funktionen der Ordnung $n + \frac{1}{2}$ und reellem Argument

   $J_{n+\frac{1}{2}}(x), Y_{n+\frac{1}{2}}(x)$

4. Funktinonen mit reeller Ordnung und reellem Argument

   $J_v(x), Y_v(x), I_v(x), K_v(x)$

5. Funktionen mit reeller Ordnung und komplexem Argument

   $J_v(z), Y_v(z), I_v(z), K_v(z)$

Der folgende Satz führt die Hankelschen Funktionen als Lösungen der Besselschen Differentialgleichung ein:

Satz 5.1.7 *Hankelsche Funktionen* Mit n ganzzahlig und

$$\begin{aligned} H_n^{(1)}(x) &:= J_n(x) + iY_n(x) \\ H_n^{(2)}(x) &:= J_n(x) - iY_n(x) \end{aligned}$$

ist $Z_n(x) = c_1 H_n^{(1)}(x) + c_2 H_n^{(2)}(x)$ eine allgemeine Lösung der Besselschen Differentialgleichung.

Die modifizierten Besselschen Funktionen erster Art der Ordnung n werden durch

$$\begin{aligned} I_n(x) &:= i^{-n} J_n(ix) \\ &= \sum_{m=0}^{\infty} \frac{1}{m!\Gamma(n+m+1)} \left(\frac{x}{2}\right)^{2m+n}, \end{aligned}$$

die modifizierten Besselschen Funktionen zweiter Art der Ordnung n durch

$$K_n(x) = \frac{\pi}{2} i^{n+1} \left(J_n(ix) + iY_n(ix)\right)$$

definiert.

5.1.4 Zylinderfunktionen und deren Anwendungen

Die Zylinderfunktionen, teilweise auch als „Besselfunktionen im weiteren Sinne“ bezeichnet, umfassen die Besselschen, Neumannschen und Hankelschen Funktionen:

Besselsche Funktionen erster Art:	$J_{\pm v}(z)$
	(Besselsche Funktionen)
Besselsche Funktionen zweiter Art:	$Y_v(z)$
	(Neumannsche Funktionen)
Besselsche Funktionen dritter Art:	$H_v^{(1)}(z), H_v^{(2)}(z)$
	(Hankelsche Funktionen)

Die Zylinderfunktionen spielen in der mathematischen Physik eine große Rolle. Sie treten u. a. bei folgenden Problemen auf (vgl. Rehwald [105]):

1. Potential in Gebieten, die von kreiszylindrischen Flächen begrenzt sind (elektrostatisches Potential, Geschwindigkeitspotential)
2. Wärmeleitung in Kreiszylindern
3. Wellenausbreitung längs zylindrischer Leiter (Sommerfeldsche Drahtleitung, Harms–Goubau–Leitung, Wendelleitung)

4. Ausbreitung von elekromagnetischen Wellen und Schallwellen in kreiszylindrischen Hohlleitern und Schwingungen in zylindrischen Resonatoren

5. Wellenausbreitung um die Erde

6. Schwingungen einer Kreismembran

7. Wechselstromwiderstand eines Kreisplattenkondensators

8. Stromverdrängung in zylindrischen Leitern

9. Streuung elektromagnetischer Wellen und Schallwellen an kugeligen Hindernissen, Streuung von Elementarteilchen an Kernen

10. Wellenfeld der Kegelantenne

11. Spektrum einer sinusförmigen frequenzmodulierten Schwingung

12. Richtdiagramme von Antennen-Kreisgruppen

13. Störung von Planetenbahnen durch andere Planeten

14. Wellenausbreitung längs Leitungen mit veränderlicher Dämpfung

Weitere Anwendungen finden sich z.B. bei Bowman [21] und Tranter [119].

5.2 Langzahl–Referenzfunktion

Beim Entwurf von (Intervall–)Funktionsimplementierungen werden (Intervall–) Langzahl–Referenzfunktionen benötigt. Mit ihrer Hilfe können z.B. Einschließungen benötigter Nullstellen berechnet werden. In leicht modifizierter Form werden sie bei der Approximationsfehlerbestimmung als „nahezu perfekte Approximation“ (siehe Kapitel 3) eingesetzt. Die Referenzfunktionen spielen eine weitere wichtige Rolle in der Testphase von Funktionsimplementierungen (vgl. Abschnitt 5.7).

Bei den Referenzfunktionen steht die Genauigkeit der berechneten Ergebnisse im Vordergrund, der Aufwand und damit das Laufzeitverhalten spielen hier nur eine untergeordnete Rolle.

Verwendet wird bei der Berechnung die Reihenentwicklung

$$\begin{aligned} J_n(x) &:= \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{2n+v} n! \Gamma(n+v+1)} x^{2n+v} \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n! \Gamma(n+v+1)} \left(\frac{x}{2}\right)^{2n+v} . \end{aligned}$$

Bei der Funktionsauswertung auf dem Rechner werden die ersten k Reihenglieder unter Verwendung einer Langzahlarithmetik berechnet und aufsummiert. Der Reihenrest wird abgeschätzt und durch das Aufaddieren eines einschließenden Intervalls sicher miterfaßt. Der Wert für den Index k und die Anzahl der benötigten Mantissenstellen der Langzahlarithmetik werden heuristisch bestimmt.

Im Rahmen dieser Arbeit wurden zwei voneinander unabhängige Referenzfunktionen implementiert. Eine Implementierung verwendet das Langzahlintervallpaket `mpi_ari` (siehe Abschnitt 1.1.2), die andere Implementierung verwendet die sogenannte Staggered Langzahlintervallarithmetik (siehe Abschnitt 1.1.1). Der Quelltext einer Referenzfunktion ist im Anhang B.4 abgedruckt.

5.3 Vorhandene Gleitkommaimplementierungen

Nach der Angabe eines typischen Algorithmus, wie er bei Gleitkommaimplementierungen zur Berechnung der Besselfunktionen eingesetzt wird, erfolgt ein Vergleich der Rechengenauigkeit mehrerer vorhandener Implementierungen.

5.3.1 Berechnungsmöglichkeit (Punktfunktionen)

Die meisten vorhandenen Implementierungen verwenden einen Algorithmus bzw. leichte Modifikationen eines Algorithmus, der bereits 1968 von Hart et. al. [43] vorgestellt wurde.

Idee des Algorithmus (Hart et. al.):

1. Berechnung von J_0, J_1, Y_0, Y_1
   (a) Unterteile $0 \leq x < \infty$ in 2 Bereiche $0 \leq x \leq D$ und $D < x < \infty$ (z.B. $D := 8$)

(b) Bereich $0 \leq x \leq D$:
J_0, $\overline{Y_0}$: Rationale Approximation in x^2
J_1, $\overline{Y_1}$: Rationale Approximation der Form $xR(x^2)$
$Y_0(x) \equiv \overline{Y_0} + \frac{2}{\pi} J_0(x) \ln(x)$
$Y_1(x) \equiv \overline{Y_1} + \frac{2}{\pi}(J_1(x) \ln(x) - \frac{1}{x}$

(c) Bereich $D < x < \infty$:
Asymtotische Entwicklungen

2. Berechnung von Y_n
Zuerst Berechnung von Y_0 und Y_1, dann Rekursionsformel
(numerisch stabil für alle n, x)

3. Berechnung von J_n
Rekursionsformel (Vorwärtsrichtung) nicht stabil für $x > n$,
Rückwärtsrekursion („Miller's Algorithmus", vgl. Abschnitt 5.4.6)

5.3.2 Vergleich vorhandener Implementierungen

In den folgenden beiden Tabellen werden Funktionswerte für die Besselfunktion $J_0(x)$ gegenübergestellt, die mittels verschiedener Computeralgebrasysteme bzw. mittels bekannter Funktionsbibliotheken berechnet wurden. In der letzten Zeile ist jeweils (mit `mpi_ari` gekennzeichnet) eine Intervalleinschließung, die mit der in Abschnitt 5.2 beschriebenen Referenzfunktion berechnet wurde, angegeben.

Beispiel: $J_0(2.405)$

Maple	$-9.055800077314\underline{61\ldots}e-5$
Mathematica	$-9.055800077300\underline{732\ldots}e-5$
Matlab	$-9.05580007730\underline{8319}e-5$
NAG C	$-9.05580007732\underline{3317}e-5$
IMSL C	$-9.055800077\underline{298337}e-5$
GNU C (2.5.8)	$-9.05580007730\underline{9275}e-5$
HP-UX C	$-9.055800077\underline{296532}e-5$
Num. Recipes	$-9.055\underline{657956455008}e-5$
`mpi_ari`	$-9.0558000773044^{70}_{68}e-5$

Als zweites Beispiel wird ein Argument verwendet, das sehr nahe an

einer Nullstelle der Funktion J_0 liegt.

Beispiel: $J_0(2.4048255576)$
$x = 40033\text{D}152\text{E}93\text{D}0\text{D}3_{\text{ HEX}} \approx 2.4048255576$

Maple	$\underline{-1.579317340\ldots}e-10$	(mit `Digits:=10`)
	$4.97201\underline{9313939\ldots}e-11$	(mit `Digits:=100`)
Mathematica	$4.97200\underline{758534\ldots}e-11$	
Matlab	$4.9720\underline{35312397402}e-11$	
NAG C	$4.9720\underline{06140135932}e-11$	
IMSL C	$4.97201\underline{7242366178}e-11$	
GNU C (2.5.8)	$4.9720\underline{06037443274}e-11$	
HP-UX C	$4.9720\underline{26400733671}e-11$	
Num. Recipes	$\underline{1.472262112692694}e-9$	
`mpi_ari`	$4.97201084117586^{2}_{0}e-11$	

Die nicht korrekten Ziffern der jeweils berechneten Ergebnisse sind unterstrichen. Die mit „Num. Recipes“ gekennzeichneten Ergebnisse wurden nach dem in Press et. al [99] angegebenen Verfahren berechnet, das Zieldatenformat dieses Verfahrens ist das IEEE single–Datenformat (Mantissenlänge 24 Bit).

Das eigentliche Problem bei Verwendung dieser im allgemeinen sehr guten Gleitkommaroutinen besteht darin, daß a priori nicht bekannt ist, wieviel Stellen des berechneten Ergebnisses korrekt sind. Insbesondere bei der Verwendung eines Computeralgebrasystems wie z.B. Maple würde man eigentlich erwarten, daß alle angezeigten Dezimalziffern korrekt sind.

5.4 Untersuchung von verschiedenen Berechnungsmöglichkeiten

5.4.1 Lösen der Differentialgleichung

Eine naheliegende Berechnungsmöglichkeit, die auch bereits von Krämer [66] vorgeschlagen wird, besteht darin, durch verifiziertes Lösen einer geeigneten Diffentialgleichung mittels AWA (siehe Lohner [77]) eine Lösung zu finden.

Die Integraldarstellung von J_n kann in der Form

$$J_n(x) = \int_0^1 \cos(x\sin(\pi t) - n\pi t)dt$$

geschrieben werden. Definiert man mit $f(x,t) := \cos(x\sin(\pi t) - n\pi t)$

$$J_n(x,y) := \int_0^y f(x,t)dt,$$

so gilt

$$J_n(x) = J_n(x,1).$$

Für beliebiges, fest vorgegebenes x ist $J_n(x,y) =: g(y)$ eine Funktion von y, welche der Differentialgleichung

$$g' = \frac{dg}{dy} = f(x,y)$$

genügt. Löst man diese Differentialgleichung verifiziert, so kann man die Funktionswerte der Besselschen Funktionen erster Gattung n-ter Ordnung für Punktargumente in Intervallschranken einschließen. Die Differentialgleichung muß dabei von 0 bis 1 mit dem Startwert $g_0 := g(0) = 0$ integriert werden.

Die benötigte Eingabedatei (versehen mit zusätzlichen Kommentaren nach dem $ -Zeichen) für das Programm AWA hat die Form

```
n = 0                              $ Besselfunktion der Ordnung n=0
x = 1                              $ Argument x=1
Pi = 4*arctan(1)                   $ Einschliessung fuer den Wert pi

F1 = cos( x*sin(Pi*T) - n*Pi*T ) $ Rechte Seite der DGl
;;

1 0 1 1e-6 24                      $      out   = 1, alle Schritte ausgegeben
                                   $  T_start   = 0, Beginn Integrationsbereich
                                   $  T_end     = 1, Ende Integrationsbereich
                                   $  h_initial = 1e-6, Anfangsschrittweite
                                   $          p = 24, Ordnung der Methode

4                                  $ Einschliesungsmethode: Schnitt von
                                   $ Intervallvektor und QR-Zerlegung
0                                  $ Ausgabe der Funktionswerte der Loesung

0                                  $ Startwert g(0) = 0

1e-16 1e-16                        $ Fehlerschranken

nnn                                $ Steuergroessen fuer AWA
```

Das Programm AWA liefert bei der Berechnung einer Einschließung für $J_0(1)$ als Endergebnis:

```
...
t =  1.000000000000000E+000
[ 7.65197686557965E-001, 7.65197686557968E-001]  1.78E-015  2.4E-015
Number of integration steps: 8
```

Bei der Rechnung in der Nähe einer Nullstelle ergibt sich folgendes Ergebnis für $J_0(2.4048255576)$:

```
...
t =  1.000000000000000E+000
[ 4.97190231190267E-011, 4.97214137910467E-011]  2.40E-015  4.9E-005
Number of integration steps: 11
```

Mit dieser Methode lassen sich zwar sichere Einschließungen der gesuchten Funktionswerte finden, jedoch erhält man in der Nähe von Nullstellen keine gute relative Genauigkeit.

Problematisch ist dieses Verfahren insbesondere für große Argumente x, da hier die Funktion $f(x,t)$ sehr stark oszilliert. Auch wächst die Laufzeit stark an.

Eine andere Einsatzmöglichkeit für das Programm AWA besteht darin, Nullstellen von Besselfunktionen zu berechnen, ohne dazu eine Funktionsimplementierung der Besselfunktionen zu benötigen. Die Grundidee ist, parallel zur Funktionswertberechnung nach der eben beschriebenen Methode einen Schritt des Intervall–Newton–Verfahrens durchzuführen. Die hierbei benötigte erste Ableitung der Besselfunktion wird mittels einer weiteren Differentialgleichung berechnet:

$$g''(y) = g'(y) = f'(x,y)$$

Mittels AWA wird also das folgende System gelöst:

$$\begin{aligned} g'(y) &= \cos(x\sin(\pi t) - n\pi t) \\ g''(y) &= -\sin(x\sin(\pi t) - n\pi t) \cdot \sin(\pi t) \end{aligned}$$

Für den Newton-Schritt gilt dann, ausgehend von einer Nullstellennäherung x:

$$x1 := x - \frac{g'(y)}{g''(y)} \tag{5.1}$$

Um einen Programmabbruch aufgrund einer möglicherweise auftretenden Division durch Null innerhalb des Integrationsbereichs $y \in [0,1]$ zu vermei-

den, wird dieser Newton-Schritt in einer leicht modifizierten Form

$$x1 := x - \frac{g'(y)}{g''(y) - y + 1} \tag{5.2}$$

verwendet. Für $y = 1$ sind die beiden Formeln (5.1) und (5.2) äquivalent.

Die Eingabedatei für AWA hat folgende Gestalt:

```
n = 0
x = 2.4048255576
Pi = 4*arctan(1)

Jn  = U1
dJn = U2

F1 =  cos( x*sin(Pi*T) - n*Pi*T )
F2 = -sin( x*sin(Pi*T) - n*Pi*T ) * sin( Pi*T )

x1 = x - Jn/( dJn - T+1 )
;
x1 Jn
;

1 0 1 1e-6 16

4
7

0
1

1e-16 1e-16

nnn
```

Nach der Durchführung eines Intervall–Newton–Schrittes erhält man

```
t =  1.000000000000000E+000
             x1: [ 2.40482555749659E+000, 2.40482555749661E+000]  5.33E-015
             Jn: [ 4.97192169691680E-011, 4.97213110741202E-011]  2.10E-015
Number of integration steps: 25
```

Die mit diesem Verfahren erzielbare Genauigkeit für die Einschließung einer Nullstelle reicht nicht aus, um anschließend eine Approximationsfunktion mit hoher (relativer) Genauigkeit auch in der Nähe der Nullstellen zu konstruieren. Hierzu werden Nullstelleneinschließungen benötigt, bei denen die Unter- und Oberschranke des einschließenden Intervalls auf mehrere Mantissenlängen des IEEE–double-Formates übereinstimmen. Diese Nullstelleneinschließungen können mit Hilfe der Referenzfunktionen (Abschnitt

5.2) gewonnen werden. Ein weiterer Nachteil der Berechnung mittels des Programms AWA ist die enorm hohe Laufzeit.

5.4.2 Integraldarstellung

Eine Berechnungsmöglichkeit für Funktionswerteinschließungen bei Besselfunktionen besteht darin, eine der Integraldarstellungen der Besselfunktionen heranzuziehen und darauf ein verifizierendes Integrationsverfahren (siehe Abschnitt 1.6) anzuwenden. Für Gleitkommanäherungen wird dies z.B. von Luke [83, Kapitel XV] vorgeschlagen.

Die Integraldarstellung

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin\theta) d\theta$$

hat den Nachteil, daß die obere Integrationsgrenze π auf dem Rechner nicht exakt darstellbar ist. Besser geeignet ist aus diesem Grund die bereits im vorhergehenden Abschnitt verwendete Form

$$J_n(x) = \int_0^1 \cos(x \sin(\pi t) - n\pi t) dt.$$

Bei einer Berechnung von Funktionseinschließungen für $J_0(x)$ mit Hilfe der Simpsonformel und der absoluten Genauigkeit 10^{-16} als Zielvorgabe erhält man folgende numerische Ergebnisse:

```
Argument x = 1.0:
[   7.65197686557963E-001,   7.65197686557970E-001 ]  6.3282712E-015

Argument x = 2.405:
[       -9.055800078E-005,       -9.055800076E-005 ]  1.3655797E-014
Argument x = 2.404825576:
[             4.971E-011,              4.973E-011 ]  1.3657331E-014
```

Der rechts angegebene Wert gibt jeweils die tatsächlich erreichte absolute Genauigkeit wieder.

Mit Hilfe dieses Verfahrens lassen sich prinzipiell Funktionswerteinschließungen für $J_n(x)$ mit beliebigen $n \in N_0$ und $x \in R$ berechnen. Allerdings nimmt die erzielbar Genauigkeit für grössere n oder für grössere x bei Rechnung mit normaler Intervallarithmetik, d.h. ohne Langszahlarithmetik, deutlich ab. Dies kann auch den folgenden drei numerischen Beispielen für $J_n(x)$ entnommen werden:

```
n = 10,  Argument x = 10.0:
[     2.074861066333E-001,      2.074861066335E-001 ]  8.4821039E-014
n = 100, Argument x = 1.0:
[               -2.4E-013,                 2.4E-013 ]  4.7324499E-013
n = 0,   Argument x = 1000:
[        2.478668615E-002,         2.478668616E-002 ]  4.0571435E-012
```

Bei der Berechnung zeigt sich weiterhin, daß die Laufzeit aufgrund des stark oszillierenden Integranden drastisch ansteigt.

5.4.3 Reihenentwicklung

Eine weitere einfache Berechnungsmöglichkeit ist die Verwendung der Reihenentwicklung

$$\begin{aligned} J_v(x) &:= \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{2n+v} n! \Gamma(n+v+1)} x^{2n+v} \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n! \Gamma(n+v+1)} \left(\frac{x}{2}\right)^{2n+v}. \end{aligned}$$

Diese wurde bei der Implementierung der Referenzfunktionen (vgl. Abschnitt 5.2) bereits eingesetzt. Ein Nachteil dieser Berechnungsmöglichkeit ist der teilweise sehr hohe Aufwand, d.h. es müssen viele Reihenglieder mitberücksichtigt werden, um ein numerisch gutes Ergebnis zu bekommen. Ein praktischer Einsatz dieser Methode ist (mit Ausnahme der Referenzfunktionen) nur für betragsmäßig sehr kleine Argumente denkbar.

5.4.4 Erzeugende Funktion (LGS)

Zu vielen speziellen Funktionen sind sogenannte „erzeugende Funktionen“ bekannt (Shrivastava/Manocha [111]). In diesem Abschnitt wird nun untersucht, inwieweit sich diese erzeugenden Funktionen zur Berechnung von Besselfunktionen einsetzen lassen (siehe auch Lohner [80]).

Eine Funktion $F(x,t)$ wird als erzeugende Funktion der Funktionen $f_n(x)$ bezeichnet, wenn eine Laurent–Reihe mit

$$F(x,t) = \sum_{n=-\infty}^{\infty} f_n(x) t^n$$

existiert. Dabei braucht die formale Reihe nicht zu konvergieren.

Grundsätzlich sind zwei Fälle zu unterscheiden:

I. $F(x,t)$ ist an der Stelle $t=0$ analytisch.

II. $F(x,t)$ ist in einer Umgebung $U_\varepsilon(0)\backslash\{0\}$ analytisch.

Im Fall I ist $f_n(x)$ offensichtlich der n-te Taylorkoeffizient von $F(x,t)$:

$$f_n(x) = \frac{1}{n!}\frac{d^n}{dt^n}F(x,t),$$

d.h. $f_n(x)$ kann einfach durch wiederholte Differentiation von $F(x,t)$ berechnet werden. Auf dem Rechner kann diese Idee mittels automatischer Differentiation (Taylorarithmetik) umgesetzt werden.

Im Fall II kann folgendermassen vorgegangen werden[4]:

Falls für $2N-1$ gegebene Werte t_j, $j=0,1,\ldots,2N-2$ die Reihenreste

$$\sum_{n=N}^{\infty} f_n(x)t_j^n \quad \text{und} \quad \sum_{n=-\infty}^{-N} f_n(x)t_j^n$$

abgeschätzt werden können und falls weiter $2N-1$ (komplexe) Intervalle M_j mit

$$\sum_{n=N}^{\infty} f_n(x)t_j^n \in M_j \quad \text{und} \quad \sum_{n=-\infty}^{-N} f_n(x)t_j^n \in M_j$$

bekannt sind, dann gilt

$$\sum_{n=-(N-1)}^{(N-1)} f_n(x)t_j^n \in F(x,t_j) - M_j, \quad j=0,1,\ldots,2N-2. \tag{5.3}$$

(5.3) ist ein (komplexes) lineares Gleichungssystem mit einem Intervallvektor als rechte Seite. Mit den Bezeichnungen

$$\begin{aligned} T &:= (t_j^n), \quad j=0,1,\ldots,2N-2, \quad n=-(N-1),\ldots,N-1,\\ f &:= (f_n(x)), \quad n=-(N-1),\ldots,N-1,\\ F &:= (F(x,t_j), \quad j=0,1,\ldots,2N-2,\\ M &:= (M_j), \quad j=0,1,\ldots,2N-2, \end{aligned}$$

kann (5.3) in der Form

$$Tf = F - M \tag{5.4}$$

[4] Der im Fall II verwendete Ansatz kann nach geringfügiger Modifikation auch im Fall I angewandt werden.

geschrieben werden. Man beachte dabei, daß T eine Punktmatrix ist und Intervalle nur auf der rechten Seite von (5.4) vorkommen. Eine optimale Einschließung des Vektors $f = (f_n(x))$ wird durch

$$f \in T^{-1}(F - M)$$

gegeben. Die Anwendung eines anderen Lösungsverfahrens für das lineare Gleichungssystem (5.4) kann zu Überschätzungen und damit zu einer gröberen Lösungseinschließung führen.

Solange die Voraussetzungen von Seite 101 (Abschätzbarkeit der Reihenreste) erfüllt werden, können die Werte t_j, $j = 0, 1, \ldots, 2N-2$ beliebig gewählt werden. Um das Gleichungssystem (5.4) einfach lösen zu können, werden im folgenden die Werte t_j als Vielfache der $(2N-1)$-ten Einheitswurzel gewählt:

$$t_j := \lambda z_j \text{ mit } z_j = \exp\left(\frac{2\pi i j}{N}\right), \quad j = 0, 1, \ldots 2N-2, \quad \lambda > 0.$$

Damit ist

$$\begin{aligned} T &= \begin{pmatrix} 1 & \lambda z_0 & \cdots & \lambda^{N-1} z_0^{N-1} \\ \vdots & \vdots & & \vdots \\ 1 & \lambda z_{N-1} & \cdots & \lambda^{N-1} z_{N-1}^{N-1} \end{pmatrix} \\ &= \begin{pmatrix} 1 & z_0 & \cdots & z_0^{N-1} \\ \vdots & \vdots & & \vdots \\ 1 & z_{N-1} & \cdots & z_{N-1}^{N-1} \end{pmatrix} \begin{pmatrix} 1 & & & 0 \\ & \lambda & & \\ & & \ddots & \\ 0 & & & \lambda^{N-1} \end{pmatrix} =: S \cdot D \end{aligned}$$

mit $D = \operatorname{diag}(1, \lambda, \ldots, \lambda^{N-1})$ diagonal und S als Matrix der diskreten Fouriertransformation. Nun kann die Lösung von (5.4) durch

$$f \in \frac{1}{N} D^{-1} S^H (F - M) \tag{5.5}$$

eingeschlossen werden. Anstelle von $\frac{1}{N} S^H$ kann eine inverse schnelle Fouriertransformation (FFT^{-1}) verwendet werden. Für $x \in \mathbb{R}$ und eine relle erzeugende Funktion F ist nur der Realteil von (5.5) relevant.

Für die Besselfunktionen ist eine erzeugende Funktion der Form (vgl. Abramowitz [1], Shrivastava/Manocha [111])

$$F(x,t) = \exp\left(\frac{1}{2} x (t - t^{-1})\right) = \sum_{-\infty}^{\infty} J_n(x) t^n$$

bekannt. Weiterhin gilt die Abschätzung (vgl. Abramowitz [1])

$$|J_n(x)| \leq \frac{1}{n!} \left|\frac{x}{2}\right|^n .$$

Mit $t_j = z_j = \exp\left(\frac{2\pi i j}{2N-1}\right)$, $j = 0, 1, \ldots, 2N-2$ findet man als Abschätzung für die Reihenreste der Laurentreihe

$$\begin{aligned}
\left| \sum_{n=-\infty}^{-N} J_n(x) t_j^n + \sum_{n=N}^{\infty} J_n(x) t_j^n \right| &\leq \sum_{n=-\infty}^{-N} |J_n(x)| + \sum_{n=N}^{\infty} |J_n(x)| \\
&= 2 \sum_{n=N}^{\infty} |J_n(x)| \\
&\leq 2 \sum_{n=N}^{\infty} \frac{1}{n!} \left|\frac{x}{2}\right|^n \\
&= 2 \frac{e^{\xi}}{n!} \left|\frac{x}{2}\right|^n \quad \text{mit } \xi \in (0, |x/2|) \\
&\leq \frac{2}{n!} e^{\left|\frac{x}{2}\right|} \left|\frac{x}{2}\right|^n .
\end{aligned}$$

Damit können die gesuchten komplexen Intervalle M_j gemäß

$$M_j := \frac{2}{n!} e^{\left|\frac{x}{2}\right|} \left|\frac{x}{2}\right|^n \cdot (1+i) \cdot [-1, 1], \quad j = 0, 1, \ldots, 2N-2$$

bestimmt werden.

Das Gleichungssystem (5.4) wird danach mit Hilfe von (5.5) gelöst.

Dieses Verfahren eignet sich besonders gut, wenn Funktionswerte $J_i(x)$, $i = 1(1)n$ mehrerer Besselfunktionen für ein festes Argument x benötigt werden. Zum Beispiel kann für $x = 1.0$ bei Vorgabe von $n = 30$ die folgende Reihe von Funktionseinschließungen berechnet werden:

```
0  [    7.6519768655796E-001,    7.6519768655798E-001 ]
1  [    4.4005058574492E-001,    4.4005058574495E-001 ]
2  [     1.149034849318E-001,     1.149034849320E-001 ]
3  [     1.956335398265E-002,     1.956335398268E-002 ]
4  [      2.47663896410E-003,      2.47663896412E-003 ]
5  [       2.4975773020E-004,       2.4975773022E-004 ]
6  [        2.093833799E-005,        2.093833802E-005 ]
7  [          1.5023258E-006,          1.5023259E-006 ]
```

```
 8  [           9.422343E-008,           9.422346E-008 ]
 9  [            5.24924E-009,            5.24926E-009 ]
10  [            2.6305E-010,             2.6307E-010 ]
11  [              1.19E-011,               1.20E-011 ]
12  [               4.9E-013,                5.1E-013 ]
13  [               1.0E-014,                2.8E-014 ]
14  [              -8.0E-015,                9.1E-015 ]
...
30  [              -9.1E-015,                8.6E-015 ]
```

Eine Berechnung für das in der Nähe einer Nullstelle von J_0 liegende Argument $x = 2.4048255576$ mit $n = 20$ liefert:

```
 0  [             4.971E-011,              4.973E-011 ]
 1  [   5.1914749731012E-001,   5.1914749731016E-001 ]
 2  [   4.3175480700433E-001,   4.3175480700437E-001 ]
 3  [    1.989999053401E-001,    1.989999053402E-001 ]
 4  [    6.474666615542E-002,    6.474666615545E-002 ]
 5  [    1.638924320185E-002,    1.638924320189E-002 ]
 6  [     3.40481847125E-003,     3.40481847129E-003 ]
 7  [      6.0068836556E-004,      6.0068836559E-004 ]
 8  [       9.216578666E-005,       9.216578669E-005 ]
 9  [        1.25172709E-005,        1.25172710E-005 ]
10  [         1.5253655E-006,         1.5253657E-006 ]
11  [          1.686022E-007,          1.686023E-007 ]
12  [          1.705343E-008,          1.705346E-008 ]
13  [            1.5900E-009,            1.5901E-009 ]
14  [             1.375E-010,             1.376E-010 ]
15  [              1.10E-011,              1.12E-011 ]
16  [               8.2E-013,               8.5E-013 ]
17  [               4.7E-014,               7.2E-014 ]
18  [              -8.8E-015,               1.7E-014 ]
19  [              -1.3E-014,               1.3E-014 ]
20  [              -1.3E-014,               1.2E-014 ]
```

Eine Implementierung unter Verwendung der komplexen Intervallarithmetik von Pascal–XSC ist im Anhang B.6 angegeben.

5.4.5 Asymptotische Entwicklungen

Zur Berechnung von Funktionseinschließungen bei Besselfunktionen für größere Argumente bietet sich die Verwendung von asymtotischen Entwick-

lungen (siehe Berg [15], Watson [121]) an. So gilt zum Beispiel

$$J_\nu(x) \approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \nu\frac{\pi}{2} - \frac{\pi}{4}\right)$$

für große Argumente x.

Die Annäherung der asymtotischen Entwicklung für $\nu = 3$ kann dem folgenden Schaubild entnommen werden.

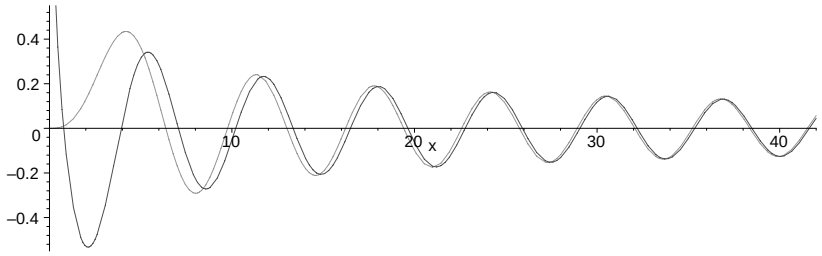


Abbildung 5.7: Besselfunktion J_3 mit asymtotischer Näherung

5.4.6 Rekursionsformeln (Drei–Term–Rekursionen)

Bei der Berechnung von Funktionswerten für Besselfunktionen höherer Ordnung spielen die Rekursionsformeln

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

$$Y_{n+1}(x) = \frac{2n}{x} Y_n(x) - Y_{n-1}(x)$$

eine wichtige Rolle. Dabei handelt es sich um Drei–Term–Rekursionen. Aus diesem Grunde erfolgen zunächst einige allgemeine Betrachtungen zu diesem Rekursionstyp und dessen Zusammenhang mit Orthogonalpolynomen.

Gegeben sei ein Skalarprodukt

$$\langle f, g\rangle := \int_a^b w(t) f(t) g(t) dt \tag{5.6}$$

mit positiver Gewichtsfunktion $w : (a,b) \to \mathbb{R}$, $w(t) > 0$ für Funktionen $f, g : [a,b] \to \mathbb{R}$. Die durch dieses Skalarprodukt induzierte Norm

$$||P|| = \sqrt{\langle P, P\rangle} = \left(\int_a^b w(t) P(t)^2 dt \right)^{1/2} < \infty$$

sei für alle Polynome $P \in \mathcal{P}_k$ und alle $k \in \mathbb{N}$ wohldefiniert und endlich. Es exisiteren dann die Momente

$$\mu_k := \int_a^b t^k w(t) dt,$$

denn aufgrund der Cauchy-Schwarzschen Ungleichung aus 1, $t^k \in \mathcal{P}_k$ folgt

$$|\mu_k| = \int_a^b t^k w(t) dt = \langle 1, t^k \rangle \leq ||1|| \cdot ||t^k|| < \infty.$$

Definition 5.4.1 *Ist $\{P_k(t)\}$ eine Folge paarweiser orthogonaler Polynome mit P_k exakt vom Grad k und*

$$\langle P_i, P_j \rangle = \int_a^b w(t) P_i(t) P_j(t) dt = \delta_{ij} \gamma_i,$$

mit $\gamma_i := ||P_i||^2 > 0$, so heißen die P_k Orthogonalpolynome über $[a,b]$ bezüglich der Gewichtsfunktion $w(t)$.

Satz 5.4.1 Zu jedem gewichteten Skalarprodukt (5.6) gibt es eindeutig bestimmte Orthogonalpolynome P_k mit führendem Koeffizienten eins. Sie genügen der Drei–Term–Rekursion

$$P_k(t) = (t + a_k) P_{k-1}(t) + b_k P_{k-2}(t) \text{ für } k = 1, 2, \ldots$$

mit den Anfangswerten $P_{-1} := 0$, $P_0 := 1$ und den Koeffizienten

$$a_k = \frac{\langle t P_{k-1}, P_{k-1} \rangle}{\langle P_{k-1}, P_{k-1} \rangle}, \quad b_k = -\frac{\langle P_{k-1}, P_{k-1} \rangle}{\langle P_{k-2}, P_{k-2} \rangle}$$

Satz 5.4.2 Das Orthogonalpolynom $P_k(t)$ hat genau k einfache (reelle) Nullstellen in (a,b).

Die reelle Drei–Term–Rekursion

$$p_k = a_k p_{k-1} + b_k p_{k-2} + c_k, \quad k = 2, 3, \ldots \tag{5.7}$$

für $p_k \in \mathbb{R}$ mit Koeffizienten a_k, b_k, $c_k \in \mathbb{R}$ und $b_k \neq 0$ für alle k läst sich auch in Rückwärtsrichtung durchführen. Es gilt dann

$$p_{k-2} = -\frac{a_k}{b_k} p_{k-1} + \frac{1}{b_k} p_k - \frac{c_k}{b_k}, \quad k = N, N-1, \ldots, 2. \tag{5.8}$$

Falls $b_k = -1$ für alle k gilt, geht (5.8) aus (5.7) durch Vertauschen von p_k und p_{k-2} hervor, eine solche Drei–Term–Rekursion heißt symmetrisch. Falls alle c_k verschwinden, heißt die Drei–Term–Rekursion homogen, andernfalls inhomogen.

Für jedes Paar p_j, p_{j+1} von Anfangswerten bestimmt (5.7) genau eine Folge $p = (p_0, p_1, \ldots) \in \mathrm{Abb}(\mathbb{N}, \mathbb{R})$. Die Lösungen $p = (p_k)$ der homogenen Drei–Term–Rekursion

$$p_k = a_k p_{k-1} + b_k p_{k-2}, \quad k = 2, 3, \ldots$$

hängen linear von den Anfangswerten p_j, p_{j+1} ab und bilden einen zweidimensionalen Unterraum

$$\mathcal{L} := \{p \in \mathrm{Abb}(\mathbb{N}, \mathbb{R}) \,|\, p_k = a_k p_{k-1} + b_k p_{k-2}, \quad k = 2, 3, \ldots\}$$

von $\mathrm{Abb}(\mathbb{N}, \mathbb{R})$. Zwei Lösungen von p, $q \in \mathcal{L}$ sind genau dann linear unabhängig, falls die sogenannten Casorati-Determinaten von p und q

$$D(k, k+1) := p_k q_{k+1} - q_k p_{k+1}$$

nicht verschwinden. Wegen $D(k, k+1) = -b_{k+1} D(k-1, k)$ und $b_k \neq 0$ verschwinden entweder alle $D(k, k+1)$ oder keine. Für symmetrische Rekursionen (d.h. $b_k = -1$) gilt $D(k, k+1) = D(0, 1)$ für alle k.

Definition 5.4.2 *Eine Lösung $p \in \mathcal{L}$ heißt rezessiv oder Minimallösung, falls für jede von p linear unabhängige Lösung $q \in \mathcal{L}$ gilt*

$$\lim_{k \to \infty} \frac{p_k}{q_k} = 0.$$

Die von p linear unabhängigen Lösungen q heißen dominant.

Die Minimallösung ist bis auf einen skalaren Faktor eindeutig bestimmt. Der skalare Faktor wird häufig durch eine Normierungsbedingung der Form

$$G_\infty := \sum_{k=0}^{\infty} m_k p_k = 1 \tag{5.9}$$

mit Gewichten m_k festgelegt.

Satz 5.4.3 Falls die Drei–Term–Rekursion symmetrisch ist und es ein $k_0 \in \mathbb{N}$ gibt, so daß $|a_k| \geq 2$ für alle $k > k_0$, dann gibt es eine Minimallösung p mit den Eigenschaften

$$|p_k| \leq \frac{1}{|a_{k+1}| - 1}|p_{k-1}| \text{ und } p_{k+1}(x) \neq 0$$

für alle $k > k_0$. Ferner gibt es für jede dominante Lösung q einen Index $k_1 \geq k_0$, so daß

$$|q_{k+1}| > (|a_{k+1}| - 1)|q_k| \text{ für } k > k_1.$$

Die Beweise der Sätze 5.4.1 bis 5.4.3 findet man bei Deuflhard/Hohmann [36].

5.4.6.1 Numerische Aspekte, Besselscher Irrgarten

Es ist bekannt, daß die Anwendung einer Drei–Term–Rekursion in einer Richtung (entweder Vorwärts- oder Rückwärtsrichtung) numerisch instabil sein kann. Für die Drei–Term–Rekursion

$$J_{k+1} = \frac{2k}{x}J_k - J_{k-1} \text{ für } k \geq 1 \tag{5.10}$$

der Besselfunktionen $J_k = J_k(x)$ entsteht hierdurch ein Verhalten, das von Deuflhard/Hohmann [36] als „Besselscher Irrgarten“ bezeichnet wird. Dieses Verhalten wird für das Argument $x = 2.13$ demonstriert, es gilt

$$\begin{aligned} J_0(2.13) &\approx 0.14960677044884 \text{ und} \\ J_1(2.13) &\approx 0.56499698056413. \end{aligned}$$

Diese beiden Werte können einer Tabelle entnommen oder mit Hilfe eines Computeralgebrasystems berechnet werden.

Mittels (5.10) werden nun die Werte $J_2, J_3, \ldots, J_{23}$ auf der Maschine (mit IEEE-double-Arithmetik) berechnet, d.h. die Rekursionsformel wird in Vorwärtsrichtung angewandt. Die numerischen Ergebnisse werden mit $\hat{J}_2, \hat{J}_3, \ldots, \hat{J}_{23}$ bezeichnet. Anschliesend wird (5.10) nach J_{k-1} aufgelöst und die Rekursion wird beginnend mit $\hat{J}_{23}$ und $\hat{J}_{22}$ in Rückwärtsrichtung durchgeführt. Die nun berechneten numerischen Werte werden mit $\bar{J}_{21}, \bar{J}_{20}, \ldots, \bar{J}_0$ bezeichnet.

Bei exakter Rechnung und bei exakt gegebenen Funktionswerten J_0 und J_1 gilt $\bar{J}_0 = J_0$. Bei Verwendung von Näherungen für J_0 und J_1 und Berechnung mittels Gleitkommaarithmetik würde man eine ungefähre Übereinstimmung erwarten, es ergibt sich aber:

$$\frac{\bar{J}_0}{J_0} \approx 10^9.$$

Der Vergleich des berechneten Wertes $\hat{J}_{23}$ mit dem tatsächlichen J_{23} ergibt

$$\frac{\hat{J}_{23}}{J_{23}} \approx 10^{27}.$$

Die Erklärung für dieses Verhalten ist, daß bei der Vorwärtsrekursion zur Berechnung von J_{k+1} sich wegen $\frac{2k}{x} J_k \approx J_{k-1}$ führende Ziffern auslöschen.

Dieses Phänomen kann mittels des folgenden Pascal-XSC Programms leicht nachvollzogen werden:

```
program irrgarten;
use i_ari;                  { Intervallarithmetik von Pascal-XSC }

var x, j0, j1,
    jk, jkm1, jkp1: interval;   { oder Datentyp real; }
    k: integer;

begin
  x := 2.13;
  j0:= 0.14960677044884;
  j1:= 0.56499698056413;

  { Rekursionformel in Vorwaertsrichtung }
  jkm1:= j0;
  jk  := j1;

  writeln(' 0: ', jkm1);
  writeln(' 1: ', jk);
  for k:= 1 to 22 do begin
    jkp1:= (2*k) / x*jk - jkm1;
    writeln(k+1:2, ': ', jkp1);
    jkm1:= jk;
    jk  := jkp1;
  end;
  writeln;
```

```
  { Rekursionsformel in Rueckwaertsrichtung }
  jkp1:= jk;
  jk  := jkm1;
  for k:= 22 downto 1 do begin
    jkm1:= (2*k) / x*jk - jkp1;
    writeln(k-1:2, ': ', jkm1);
    jkp1:= jk;
    jk  := jkm1;
  end;
end.
```

Bei Anwendung naiver Intervallrechnung erhält man die nachfolgenden Ergebnisausgaben, an denen die Auswirkungen der Auslöschung führender Ziffern sehr schön sichtbar sind:

```
 0: [  1.496067704488399E-001,  1.496067704488400E-001 ]
 1: [  5.649969805641299E-001,  5.649969805641300E-001 ]
 2: [  3.809068263249908E-001,  3.809068263249911E-001 ]
...
21: [               1.9E+003,               2.3E+003 ]
22: [               3.7E+004,               4.4E+004 ]
23: [               7.8E+005,               9.0E+005 ]

21: [              -1.2E+005,               1.3E+005 ]
20: [              -2.4E+006,               2.4E+006 ]
19: [              -4.5E+007,               4.5E+007 ]
...
 2: [              -1.3E+024,               1.3E+024 ]
 1: [              -2.8E+024,               2.8E+024 ]
 0: [              -3.9E+024,               3.9E+024 ]
```

5.4.6.2 Miller–Algorithmus (Rückwärtsrekursion)

Wie am Beispiel des Besselschen Irrgartens deutlich wird, ist die direkte Anwendung der Drei–Term–Rekursion zur numerischen Berechnung von Minimallösungen nicht geeignet. Eine Möglichkeit, wie die Drei–Term–Rekursion dennoch bei der numerischen Berechnung verwendet werden kann, wurde von Miller [92] im Jahre 1952 aufgezeigt.

Die Grundidee dabei ist, die Rekursionsformel für die Besselfunktionen in der numerisch stabilen Rückwärtsrichtung zu verwenden und zusätzlich

die Normierungsbedingung (5.9), approximiert durch die Teilsummen

$$G_n := \sum_{k=0}^{n} m_k p_k$$

anzuwenden.

Der Algorithmus kann folgendermassen angegeben werden:

Algorithmus 5.1: Miller (a_k, b_k, N) Algorithmus von Miller
1. Wähle Abbruchindex $n > N$ 2. Setze $\hat{p}_{n+1}^{(n)} := 0$, $\hat{p}_n^{(n)} := 1$ 3. Berechne $\hat{p}_{n-1}^{(n)}, \ldots, \hat{p}_0^{(n)}$ aus $$\hat{p}_{k-2}^{(n)} = \frac{1}{b_k}(\hat{p}_k^{(n)} - a_k \hat{p}_{k-1}^{(n)})$$ für $k = n + 1(-1)2$ 4. Berechne $\hat{G}_n := \sum_{k=0}^{n} m_k \hat{p}_k$ 5. Normiere $p_k^{(n)} := \hat{p}_k^{(n)} / \hat{G}_n$ 6. Wiederhole Schritte 1 bis 5 solange für wachsende $n = n_1, n_2, \ldots$ bis $\lvert p_N^{(n_i)} - p_N^{(n_{i-1})} \rvert \leq \varepsilon p_N^{n_i}$ für vorgegebenes $\varepsilon > 0$

Die Konvergenz des Algorithmus Miller liefert der

Satz 5.4.4 Sei $p \in \mathcal{L}$ eine Minimallösung einer homogenen Drei–Term–Rekursion, die der Normierungsbedingung

$$\sum_{k=0}^{\infty} m_k p_k = 1$$

genüge. Zusätzlich gebe es eine dominante Lösung $q \in \mathcal{L}$, so daß

$$\lim_{n\to\infty} \frac{p_{n+1}}{q_{n+1}} \sum_{k=0}^{n} m_k q_k = 0.$$

Dann konvergiert die Folge der Miller–Approximationen $p_N^{(n)}$ gegen p_N, $\lim_{n\to\infty} p_N^{(n)} = p_N$.

Beweis: siehe Deuflhard/Hohmann [36].

Die Anwendung der Idee des Miller–Algorithmuses wird beispielhaft an der Berechnung der Funktionswerte $J_k(x)$ mit $k = 0, 1, 2, \ldots, 6$ für das Argument $x = 1$ aufgezeigt. Hierbei wird die Rekursionformel

$$J_{n-1}(x) = \frac{2n}{x} J_n(x) - J_{n+1}(x)$$

und die Normierungsbedingung

$$1 = J_0(x) + 2 \sum_{n=1}^{\infty} J_{2n}(x)$$

verwendet.

Zuerst wird die Rückwärtsrekursion beginnend mit den Startwerten 0 und 1 für J_{10}^* bzw. J_9^* durchgeführt. Parallel dazu wird der für die Normierungsbedingung benötigte Wert s aufsummiert.

					Summation	
J_{10}^*	:=	0			0	
J_9^*	:=	1			0	
J_8^*	=	$18 \cdot 1 - 0$	=	18	36	
J_7^*	=	$16 \cdot 18 - 1$	=	287	36	
J_6^*	=	$14 \cdot 287 - 18$	=	4000	8036	
J_5^*	=	$12 \cdot 4000 - 287$	=	47713	8036	
J_4^*	=	$10 \cdot 47713 - 4000$	=	473130	954296	
J_3^*	=	$8 \cdot 473130 - 47713$	=	3737327	954296	
J_2^*	=	$6 \cdot 3737327 - 473130$	=	21950832	44855960	
J_1^*	=	$4 \cdot 21950832 - 3737327$	=	84066001	44855960	
J_0^*	=	$2 \cdot 84066001 - 21950832$	=	146181170	191037130	$=: s$

In einem zweiten Schritt wird nun die Normierungsbedingung angewandt. Um die mit dem Miller–Algorthmus berechneten Werte vergleichen zu können, sind in der rechten Spalte Einschließungen[5] des tatsächlichen

[5] Berechnet mit Hilfe der Referenzfunktion.

Funktionswertes angegeben.

					Einschließung
J_{10}	=	J_{10}^*/s	=	0	$[2.630615123_6^7e-10]$
J_9	=	J_9^*/s	=	$5.234584502e-9$	$[5.2492501_{799}^{800}e-9]$
J_8	=	J_8^*/s	=	$9.422252104e-8$	$[9.42234417_6^7e-8]$
J_7	=	J_7^*/s	=	$1.502325752e-6$	$[1.50232581_4^7e-6]$
J_6	=	J_6^*/s	=	$2.093833801e-5$	$[2.093833800_2^3e-5]$
J_5	=	J_5^*/s	=	$2.497577303e-4$	$[2.497577302_1^2e-4]$
J_4	=	J_4^*/s	=	$2.476638965e-3$	$[2.476638964_1^2e-3]$
J_3	=	J_3^*/s	=	$1.956335399e-2$	$[1.956335398_2^3e-2]$
J_2	=	J_2^*/s	=	0.114903485	$[0.1149034_{49}^{50}]$
J_1	=	J_1^*/s	=	0.440050586	$[0.44005058_7^8]$
J_0	=	J_0^*/s	=	0.765197687	$[0.76519768_5^6]$

Eine Pascal–XSC Implementierung eines Programms zur Umsetzung des Miller–Algorithmus ist im Anhang B.5 angegeben. Dieses Programm liefert für das Argument $x = 2.13$ die folgenden Ausgaben:

```
x = 2.13
n = 16

 17   0.000000000000000E+000
 16   1.219310061993320E-013
 15   1.831827323182452E-012
 14   2.567845382735633E-011
 13   3.357253121913609E-010
 12   4.072377000151697E-009
 11   4.555021271768692E-008
 10   4.663993036520136E-007
  9   4.333785971808263E-006
  8   3.615714412007980E-005
  7   2.672691745546127E-004
  6   1.720541655769394E-003
  5   9.425923252315213E-003
  4   4.253261915322222E-002
  3   1.503210031447635E-001
  2   3.809068263249850E-001
  1   5.649969805641285E-001
  0   1.496067704488445E-001
```

Eine Modifikation des Algorithmus von Miller besteht darin, die Dar-

stellung von $\frac{J_n(x)}{J_{n-1}(x)}$ in Form eines Kettenbruchs (Abramowitz [1]) als

$$\frac{J_n(x)}{J_{n-1}(x)} = \frac{x}{2n-}\ \frac{x^2}{2(n+1)-}\ \frac{x^2}{2(n+2)-}\ \cdots \tag{5.11}$$

auszunutzen.

Die Idee wird wieder am Beispiel der Berechnung der Funktionswerte $J_k(x)$ mit $k = 0, 1, 2, \ldots, 6$ für das Argument $x = 1$ gezeigt.

Zur Durchführung der Rückwärtsrekursion wird diesmal nur J^*_{10} auf den Wert 1 gesetzt. Der Startwert J^*_9 wird mit Hilfe des (abgebrochenen) Kettenbruchs 5.11 berechnet:

$$\frac{J_6}{J_5} \approx \frac{1}{12-}\ \frac{1}{14-}\ \frac{1}{16-}\ \frac{1}{18-}\ \frac{1}{20} = 8.383459437e-2 =: k$$

Es ergeben sich damit die Werte

$$\begin{array}{lcll}
J^*_6 & := & 1 & = & 1 \\
J^*_5 & = & J_6/k & = & 11.92825 \\
J^*_4 & = & 10 \cdot 11.92825 - 1 & = & 118.2825 \\
J^*_3 & = & 8 \cdot 118.2825 - 11.92825 & = & 934.33175 \\
J^*_2 & = & 6 \cdot 934.33175 - 118.2825 & = & 5487.708 \\
J^*_1 & = & 4 \cdot 5487.708 - 934.33175 & = & 21016.50025 \\
J^*_0 & = & 2 \cdot 21016.50025 - 5487.708 & = & 36545.2925
\end{array}$$

Nun wird der Quotient aus einem bekannten bzw. zuvor berechnetem Näherungswert für $J_0(x)$ und J^*_0 gebildet:

$$J_0(1.0) \approx 0.765197687 =: c \cdot J^*_0$$

$$\Longrightarrow c = \frac{0.765197687}{36545.2925} = 2.093833801e-5$$

Mit dem so ermittelten Wert c wird nun eine Normierung durchgeführt. Zum Vergleich sind in der rechten Spalte wieder Einschließungen des tatsächlichen Funktionswertes angegeben.

$$\begin{array}{lcllll}
 & & & & & \text{Einschließung} \\
J_6 & = & c \cdot J^*_6 & = & 2.093833801e-5 & [2.093833800^{3}_{2}e-5] \\
J_5 & = & c \cdot J^*_5 & = & 2.497577304e-4 & [2.497577302^{2}_{1}e-4] \\
J_4 & = & c \cdot J^*_4 & = & 2.476638966e-3 & [2.476638964^{2}_{1}e-3] \\
J_3 & = & c \cdot J^*_3 & = & 1.956335399e-2 & [1.956335398^{3}_{2}e-2] \\
J_2 & = & c \cdot J^*_2 & = & 0.114903485 & [0.1149034\,8^{50}_{49}] \\
J_1 & = & c \cdot J^*_1 & = & 0.440050586 & [0.440050585^{8}_{7}] \\
J_0 & = & c \cdot J^*_0 & = & 0.765197687 & [0.765197686^{6}_{5}]
\end{array}$$

5.4.6.3 Lineares Gleichungssytem mit Bandmatrix

Die homogene Drei–Term–Rekursion

$$p_k = a_k p_{k-1} + b_k p_{k-2}, \quad k = 2(1)N$$

mit den Startwerten $p_0 := \alpha$ und $p_1 := \beta$ kann als Gleichungssystem

$$\underbrace{\begin{pmatrix} 1 & & & & 0 \\ 0 & 1 & & & \\ -b_2 & -a_2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ 0 & & -b_N & -a_N & 1 \end{pmatrix}}_{:= L} \underbrace{\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ \vdots \\ p_N \end{pmatrix}}_{:= p} = \underbrace{\begin{pmatrix} \alpha \\ \beta \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{:= r}$$

mit einer Bandmatrix L geschrieben werden. Die Berechnung von $p^T = (p_0, \ldots, p_N)$ erfolgt nun durch Lösen des $(N+1)$-dimensionalen linearen Gleichungssystems $Lp = r$.

Angewandt auf die Rekursionsformel der Besselfunktionen

$$J_k(x) = \frac{2k}{x} J_{k-1}(x) - J_{k-2}(x), \quad k = 2(1)N$$

ergibt sich das lineare Gleichungssystem

$$\underbrace{\begin{pmatrix} 1 & & & & 0 \\ 0 & 1 & & & \\ 1 & \frac{-2\cdot 1}{x} & 1 & & \\ & \ddots & \ddots & \ddots & \\ 0 & & 1 & \frac{-2\cdot(N-1)}{x} & 1 \end{pmatrix}}_{= L} \underbrace{\begin{pmatrix} J_0 \\ J_1 \\ J_2 \\ \vdots \\ J_N \end{pmatrix}}_{= p} = \underbrace{\begin{pmatrix} J_0(x) \\ J_1(x) \\ 0 \\ \vdots \\ 0 \end{pmatrix}}_{= r} .$$

Intervalleinschließungen des gesuchten Lösungsvektors p können mit dem in Abschnitt 1.5 vorgestellten Bandlöser berechnet werden.

5.4.7 Polynomiale Darstellung

Ausgehend von der je nach Rekursionsrichtung numerisch instabilen Rekursionsformel

$$J_{n-1}(x) = \frac{2n}{x} J_n(x) - J_{n+1}(x)$$

für die Besselfunktionen wird nun versucht, eine für numerische Zwecke besser geeignete Berechnungsmöglichkeit herzuleiten.

Hierzu werden Polynomfolgen $p = (p_i)$, $i = 1, 2, \ldots$ und $q = (q_i)$, $i = 0, 1, 2, \ldots$ rekursiv definiert:

$$\begin{aligned} p_1(x) &:= x \\ p_2(x) &:= 2x^2 - 1 \\ p_n(x) &:= n \cdot x \cdot p_{n-1}(x) - p_{n-2}(x) \\ \text{und} & \\ q_0(x) &:= 1 \\ q_1(x) &:= 2x \\ q_n(x) &:= (n+1) \cdot x \cdot p_{n-1}(x) - p_{n-2}(x) \end{aligned}$$

Damit kann der folgende Satz[6] formuliert werden:

Satz 5.4.5 Es gilt

$$J_{n+1}(x) = p_n(\frac{2}{x}) \cdot J_1(x) - q_{n-1}(\frac{2}{x}) \cdot J_0(x), \quad n \geq 1.$$

Die Berechnung des Wertes $J_{n+1}(x)$ kann damit im wesentlichen auf die im allgemeinen unkritische Auswertung der beiden Polynome $p_n(2/x)$ und $q_{n-1}(2/x)$ (z.B. mittels Horner-Schema) und Multiplikation mit $J_1(x)$ bzw. $J_0(x)$ zurückgeführt werden. Die einzige bezüglich Auslöschung kritische Operation ist die anschließend auszuführende Subtraktion der beiden Werte $p_n(2/x) \cdot J_1(x)$ und $q_{n-1}(2/x) \cdot J_0(x)$, diese kann dann auf dem Rechner gezielt mit höherer Genauigkeit oder unter Verwendung des exakten Skalarproduktausdrucks ausgeführt werden.

Bei der praktischen Umsetzung bietet es sich an, die Koeffizienten der Polynome p_i, $i = 1, 2, \ldots$ und q_i, $i = 0, 1, 2, \ldots$ für kleinere Indizes i in einer Tabelle abzuspeichern. Die hierzu benötigten Koeffizientenvektoren $(p_{n,0}, p_{n,1}, p_{n,2}, \ldots, p_{n,n})$ für das Polynom $p_n(x) = \sum_{i=0}^{n} p_{n,i} x^i$ vom Grad n bzw. $(q_{n-1,0}, q_{n-1,1}, q_{n-1,2}, \ldots, q_{n-1,n-1})$ für das Polynom $q_{n-1}(x) = \sum_{i=0}^{n-1} q_{n-1,i} x^i$ vom Grad $n-1$ finden sich in den Tabellen 5.1 und 5.2. Für

[6] Eine prinzipiell ähnliche Darstellung für J_n findet sich bei Watson [121, Kapitel IX] unter dem Begriff Lommelsche Polynome.

größere Indizes i können die benötigten Polynomkoeffizienten mittels einer einfachen Polynomarithmetik zur Laufzeit berechnet werden. Eine Implementierung einer solchen Arithmetik in Pascal–XSC ist im Anhang B.7 angegeben.

Polynom	zugehöriger Koeffizientenvektor
p_1	$(0, 1)$
p_2	$(-1, 0, 2)$
p_3	$(0, -4, 0, 6)$
p_4	$(1, 0, -18, 0, 24)$
p_5	$(0, 9, 0, -96, 0, 120)$
p_6	$(-1, 0, 72, 0, -600, 0, 720)$
p_7	$(0, -16, 0, 600, 0, -4320, 0, 5040)$
p_8	$(1, 0, -200, 0, 5400, 0, -35280, 0, 40320)$
p_9	$(0, 25, 0, -2400, 0, 52920, 0, -322560, 0, 362880)$
p_{10}	$(-1, 0, 450, 0, -29400, 0, 564480, 0, -3265920, 0, 3628800)$

Tabelle 5.1: Polynomiale Darstellung: Koeffizienten für $p_n(x)$

Polynom	zugehöriger Koeffizientenvektor
q_0	(1)
q_1	$(0, 2)$
q_2	$(-1, 0, 6)$
q_3	$(0, -6, 0, 24)$
q_4	$(1, 0, -36, 0, 120)$
q_5	$(0, 12, 0, -240, 0, 720)$
q_6	$(-1, 0, 120, 0, -1800, 0, 5040)$
q_7	$(0, -20, 0, 1200, 0, -15120, 0, 40320)$
q_8	$(1, 0, -300, 0, 12600, 0, -141120, 0, 362880)$
q_9	$(0, 30, 0, -4200, 0, 141120, 0, -1451520, 0, 3628800)$

Tabelle 5.2: Polynomiale Darstellung: Koeffizienten für $q_{n-1}(x)$

Mit dem Verfahren der polynomialen Darstellung können z.B. die folgenden Einschließungen für die Funktionswerte $J_k(1.0)$, $k = 0(1)10$ berechnet werden:

```
 J0 = [  7.651976865579664E-001,  7.651976865579667E-001 ]
 J1 = [  4.400505857449334E-001,  4.400505857449336E-001 ]
 J2 = [  1.149034849319004E-001,  1.149034849319005E-001 ]
 J3 = [  1.956335398266840E-002,  1.956335398266841E-002 ]
 J4 = [  2.476638964109954E-003,  2.476638964109956E-003 ]
 J5 = [  2.497577302112343E-004,  2.497577302112345E-004 ]
 J6 = [  2.093833800238926E-005,  2.093833800238928E-005 ]
 J7 = [  1.502325817436808E-006,  1.502325817436809E-006 ]
 J8 = [  9.422344172604500E-008,  9.422344172604502E-008 ]
 J9 = [  5.249250179911874E-009,  5.249250179911876E-009 ]
J10 = [    2.6306151236874E-010,    2.6306151236875E-010 ]
```

Durch den gezielten Einsatz des exakten Skalarprodukts bzw. einer höher genauen Arithmetik erhält man hier wesentlich engere Einschließungen (vgl. mit dem numerischen Beispiel in Abschnitt 5.4.4).

5.5 Nullstellen und Extremalstellen der Besselfunktionen

Insbesondere zur Konstruktion von Approximationsfunktionen mit hoher relativer Genauigkeit auch im Bereich der Funktionsnullstellen werden Langzahl–Intervalleinschließungen der entsprechenden Nullstellen benötigt.

Eine Möglichkeit zur Berechnung von Einschließungen der Nullstellen ist die Anwendung eines (Intervall–) Bisektionsverfahrens (vgl. Vrahatis [118]) zusammen mit den (Langzahl–) Referenzfunktionen (Abschnitt 5.2). Der Vorteil hierbei ist, daß keine Funktionsableitungen benötigt werden.

Eine weitere Möglichkeit ist der Einsatz eines langzahlmäßigen Intervall–Newton–Verfahrens (Abschnitt 1.4). Zur Berechnung der Nullstellen von J_0 wird die Ableitung J_0' verwendet. Nach dem Differentiationssatz 5.1.3 gilt

$$J_0' = -J_1,$$

d.h. die benötigten Werte der Ableitungsfunktion können über die Referenzfunktion berechnet werden. Bei der Nullstelleneinschließung von J_n, $n \geq 1$ kann auf den Rekursionssatz 5.1.4 zurückgegriffen werden, es gilt

$$J_n' = \frac{1}{2}(J_{n-1} - J_{n+1}), \quad n \geq 1.$$

Die Berechnung ist damit ebenfalls über die Referenzfunktion (allerdings mit zwei Funktionsauswertungen) möglich.

Welche der beiden Möglichkeiten effizienter ist, hängt von der Anzahl der jeweils benötigten Auswertungen der Referenzfunktion ab.

Langzahl–Intervalleinschließungen für die ersten sechs positiven Nullstellen von J_0 werden in der folgenden Tabelle 5.3 exemplarisch angegeben. Nach dem Nullstellensatz 5.1.6 handelt es sich bei allen positiven Nullstellen um einfach Nullstellen.

Alle Nullstellen von $J_0(x)$ im offenen Intervall $(0, 100)$ sind, gerundet auf die Genauigkeit des IEEE double–Formates, in Tabelle 5.4 aufgelistet.

Die Nullstellen von $J_1(x)$ im offenen Intervall $(0, 100)$ sind in Tabelle 5.5 angegeben. Es handelt sich hierbei gleichzeitig um alle Extremalstellen der Funktion $J_0(x)$ im Intervall $(0, 100)$. Die notwendige Bedingung für die Extremalstellen folgt aus $J_0'(x) = -J_1(x) = 0$. Die hinreichende Bedingung folgt aus dem Nullstellensatz 5.1.6.

Nullstelle	Langzahl–Intervalleinschließung
$j_{0,1}$	$2.40482555769577276862163187932645464312424490914596713570699909059676583867719402920444_3^4$
$j_{0,2}$	$5.52007811028631064959660411281302742522186547878290985375755203814429082919372544309362_1^2$
$j_{0,3}$	$8.65372791291101221695419871266094668556579523127535561889147658302259995665099877226732_2^3$
$j_{0,4}$	$11.79153443901428161374304491192545892202292469969544670325051087905164651179274511097774_3^4$
$j_{0,5}$	$14.93091770848778594776259399738868220791585011563302815877417321883519336393209342637676_6^7$
$j_{0,6}$	$18.07106396791092254314788297561817656024898674700132608642331463528382056239377834254921_1^2$

Tabelle 5.3: Einschließungen der ersten sechs positiven Nullstellen von $J_0(x)$

Nullstelle	Einschließung
$j_{0,1}$	$2.404825557695773_{2}^{3}$
$j_{0,2}$	$5.520078110286311_{0}^{1}$
$j_{0,3}$	$8.653727912911013_{2}^{3}$
$j_{0,4}$	$11.791534439014299_{8}^{9}$
$j_{0,5}$	$14.930917708487799_{8}^{9}$
$j_{0,6}$	$18.071063967910923_{2}^{3}$
$j_{0,7}$	$21.211636629879266_{5}^{6}$
$j_{0,8}$	$24.352471530749301_{0}^{1}$
$j_{0,9}$	$27.493479132040266_{5}^{6}$
$j_{0,10}$	$30.634606468431978_{7}^{8}$
$j_{0,11}$	$33.775820213573567_{6}^{7}$
$j_{0,12}$	$36.917098353664045_{4}^{5}$
$j_{0,13}$	$40.058425764628243_{3}^{4}$
$j_{0,14}$	$43.199791713176743_{3}^{4}$
$j_{0,15}$	$46.341188371661782_{1}^{2}$
$j_{0,16}$	$49.482609897397782_{1}^{2}$
$j_{0,17}$	$52.624051841115_{499}^{500}$
$j_{0,18}$	$55.765510755019978_{7}^{8}$
$j_{0,19}$	$58.906983926080945_{4}^{5}$
$j_{0,20}$	$62.048469190227166_{6}^{7}$
$j_{0,21}$	$65.189964800206857_{6}^{7}$
$j_{0,22}$	$68.331469329856_{79}^{80}$
$j_{0,23}$	$71.472981603593744_{3}^{4}$
$j_{0,24}$	$74.614500643701844_{3}^{4}$
$j_{0,25}$	$77.756025630388066_{5}^{6}$
$j_{0,26}$	$80.897555871137643_{2}^{3}$
$j_{0,27}$	$84.039090776938_{19}^{20}$
$j_{0,28}$	$87.180629843641166_{5}^{6}$
$j_{0,29}$	$90.322172637210449_{8}^{9}$
$j_{0,30}$	$93.463718781944678_{7}^{8}$
$j_{0,31}$	$96.605267950996267_{6}^{7}$
$j_{0,32}$	$99.746819858680_{59}^{60}$

Tabelle 5.4: Einschließungen der Nullstellen von $J_0(x)$ im Bereich $0 < x < 100$

Nullstelle	Einschließung
$j_{1,1}$	3.83170597020751^{3}_{2}
$j_{1,2}$	$7.015586669815619^{9}_{8}$
$j_{1,3}$	$10.17346813506273^{3}_{2}$
$j_{1,4}$	$13.32369193631423^{3}_{2}$
$j_{1,5}$	$16.47063005087764^{4}_{3}$
$j_{1,6}$	$19.61585851046825^{5}_{4}$
$j_{1,7}$	$22.76008438059278^{8}_{7}$
$j_{1,8}$	$25.90367208761839^{9}_{8}$
$j_{1,9}$	$29.04682853491686^{6}_{5}$
$j_{1,10}$	$32.18967991097441^{1}_{0}$
$j_{1,11}$	$35.33230755008387^{7}_{6}$
$j_{1,12}$	$38.47476623477162^{2}_{1}$
$j_{1,13}$	$41.61709421281446^{6}_{5}$
$j_{1,14}$	$44.75931899765283^{3}_{2}$
$j_{1,15}$	$47.90146088718545^{5}_{4}$
$j_{1,16}$	$51.04353518357151^{1}_{0}$
$j_{1,17}$	$54.18555364106133^{3}_{2}$
$j_{1,18}$	$57.32752543790102^{2}_{1}$
$j_{1,19}$	$60.46945784534750^{50}_{49}$
$j_{1,20}$	$63.61135669848124^{4}_{3}$
$j_{1,21}$	$66.75322673409850^{50}_{49}$
$j_{1,22}$	$69.89507183749578^{8}_{7}$
$j_{1,23}$	$73.03689522557384^{4}_{3}$
$j_{1,24}$	$76.17869958464146^{6}_{5}$
$j_{1,25}$	$79.32048717547630^{30}_{29}$
$j_{1,26}$	$82.46225991437356^{6}_{5}$
$j_{1,27}$	$85.60401943635024^{4}_{3}$
$j_{1,28}$	$88.74576714492631^{1}_{0}$
$j_{1,29}$	$91.88750425169499^{9}_{8}$
$j_{1,30}$	$95.02923180804470^{70}_{69}$
$j_{1,31}$	$98.17095073079079^{9}_{8}$

Tabelle 5.5: Einschließungen der Nullstellen von $J_1(x)$ im Bereich $0 < x < 100$

5.6 Die Funktion J_0

Das Vorgehen beim Entwurf einer Funktionsroutine mit sicheren Fehlerschranken für die Besselfunktion J_0 folgt den in Abschnitt 4.2 angegebenen 7 Schritten.

Für die in Schritt 1 geforderte nahezu perfekte Intervallapproximation kann die erstellte Referenzfunktion (Abschnitt 5.2) verwendet werden, wenn die zu implementierende Funktion J_0 direkt approximiert wird. Um auch in der Nähe der Funktionsnullstellen eine gute relative Fehlerschranke zu erreichen, ist es nötig, eine geeignete Hilfsfunktion H zu approximieren (vgl. Beschreibung der Grundlagen im Kapitel 3).

Der auf dem Rechner maximal mögliche und zulässige Argumentbereich [`-maxreal`, `maxreal`] kann zunächst auf das Intervall [0, `maxreal`] eingeschränkt werden. Funktionswerteinschließungen für negative Argumente können später im Algorithmus mit Hilfe der Symmetrieeigenschaft

$$J_n(-x) = (-1)^n J_n(x) \quad \text{bzw.} \quad J_0(-x) = J_0(x)$$

bereitgestellt werden. Das Intervall [0, `maxreal`] wird in mehrere Teilintervalle zerlegt, die getrennt weiteruntersucht werden.

Im folgenden wird nur die Approximation der Funktion J_0 im Teilintervall $[0, 6]$ beschrieben. In diesem Intervall besitzt die Funktion J_0 zwei Nullstellen $n_1 := j_{0,1}$ und $n_2 := j_{0,2}$. Die zu approximierende Hilfsfunktion wird deshalb folgendermassen definiert:

$$H(x) := \frac{J_0(x)}{(x - n_1)(x - n_2)} \quad \text{mit } x \neq n_1,\ x \neq n_2$$

Die Berechnung der nahezu perfekten Intervallapproximation der Hilfsfunktion H erfolgt mit dem Modul `mpitaylor`, das eine Taylorarithmetik (vgl. Abschnitt 1.2) unter Verwendung der Langzahlintervallarithmetik `mpi_ari` (vgl. Abschnitt 1.1.2) zur Verfügung stellt.

Im Schritt 2 werden Einschließungen der Nullstellen von J_0, d.h. unter anderem von n_1 und n_2, benötigt, diese wurden bereits in Abschnitt 5.5 berechnet.

Da bei der Berechnung der Besselfunktion J_0 keine Argumentreduktion durchgeführt wird, entfallen die Schritte 3 und 6.

Die Bestimmung einer geeigneten (Best-)Approximation im Schritt 4 erfolgt mit Hilfe des Computeralgebrasystems Maple. Hierzu wird die Maple–Funktion `minimax` eingesetzt, die Berechnung findet unter Verwendung des Remez–Algorithmus statt. Mit den Vorgaben Polynomgrad des Zählers := 5

und Polynomgrad des Nenners := 3 lassen sich z.B. folgende Polynomkoeffizienten für eine rationale Approximation von

$$H(x) \approx \frac{P(x^2)}{Q(x^2)}, \quad x \in [0,6]$$

finden:

```
Polynomkoeffizienten des Zaehlerpolynoms:
3F73AD6441149448    4.804031007812411E-003
BF2628A7FFCC08A7   -1.690583302647642E-004
3EC331AC0C0DCE5B    2.288106856368657E-006
BE508ABAA9A05970   -1.540585490435812E-008
3DCD648FF82F0E93    5.346523139404624E-011
BD365D601304B99A   -7.945554339748063E-014
Polynomkoeffizienten des Nennerpolynoms:
3FEB171C2438BF25    8.465710360960413E-001
3F7F78E1B56F0909    7.683641123494186E-003
3EFF0916CBD583A7    2.959776315003865E-005
3E6B355F442457F6    5.067975241046154E-008
```

Die angegebenen Werte (jeweils als exakte hexadezimale Zahl und als gerundete Dezimalzahl) sind dabei bereits in das IEEE double–Zahlformat gerundet worden.

Zur Abschätzung des Approximationsfehlers im Schritt 5 wird das in Kapitel 3 erläuterte Verfahren eingesetzt. Hierbei wird, analog wie im Anwendungsbeispiel von Kapitel 3, die äquivalente Approximation

$$H(\sqrt{y}) \approx \frac{P(y)}{Q(y)} \quad \text{mit } y \in [0,36]$$

untersucht.

Die Abbildung 5.8 zeigt den Fehlerverlauf der Approximation. Man beachte, daß die Funktionswerte wieder mit dem Faktor 10^{-16} skaliert sind.

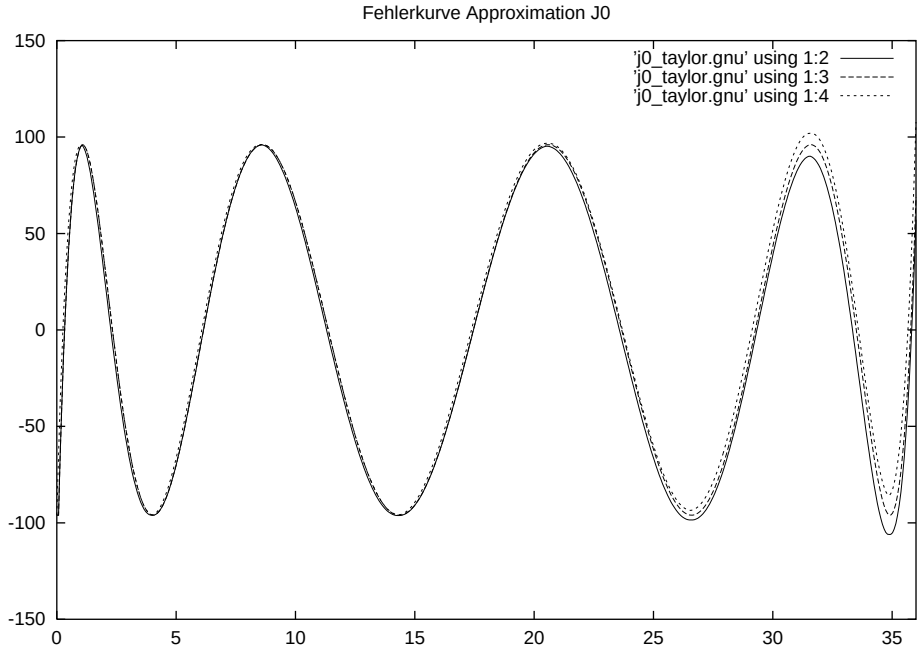


Abbildung 5.8: Einschließung der Fehlerkurve

Dem folgenden Ausschnitt des Ausgabeprotokolls des Programms kann die maximale absolute Fehlerschranke für den Approximationsfehler entnommen werden:

```
Approximationsintervall = 0.000000000000000E+000  3.600000000000000E+001
Funktionsauswertung ueber Gesamtintervall:
[               -3.4E-003,                 5.7E-003 ]
Entwicklungsstelle x0 =  0.000000000000000E+000
Mini: -9.604031156848087E-015   Maxi: -9.604030956848079E-015
Mini: -9.604031156848087E-015   Maxi:  9.603385364887501E-015
Gewaehlte Anzahl von Teilintervallen: 10000000
Breite eines Teilintervalls:  3.600000000000000E-006
Endergebnis:
Mini:   -1.060321344292708E-014      Maxi:  1.079389102883999E-014
MiniUb: -8.604023900294400E-015    MaxiLb:  8.412734111818050E-015
```

Die im Schritt 7 geforderte endgültige Gesamtfehlerschranke für das implementierte Verfahren zur Berechnung von J_0 wird unter Einsatz des Fehlerkalküls aus Kapitel 2 ermittelt.

Um die entwickelte Funktion in eine bestehende (Intervall-) Funktionsbibliothek (wie z.B. `FI_LIB` [51], [52]) integrieren zu können, erfolgte die Implementierung in der Programmiersprache ANSI–C. Hierdurch wird auch die Übertragbarkeit der Routine auf verschiedene Rechnerplattformen erleichtert. Einen Auszug aus dem Quelltext der ANSI–C Implementierung ist im Anhang B.8 angegeben.

5.7 Test der Implementierungen

Beim Testen von (Intervall-)Funktionsimplementierungen geht es in erster Linie darum, Programmierfehler auszuschließen. Hierzu wurden zwei getrennte Langzahl–Referenzfunktionen unter Verwendung von voneinander unabhängigen Langzahlintervallpaketen entwickelt. Ein erster Testschritt ist nun der automatisierte Ergebnisvergleich der zu testenden Funktion mit den beiden Referenzfunktionen für möglichst viele Zufallsargumente. Ein weiterer Testschritt ist der Vergleich für eine begrenzte Anzahl vorab ausgewählter Eingangsargumente. Diese Eingangsargumente werden dabei so bestimmt, daß alle Fallunterscheidungen der verwendeten Algorithmen mindestens einmal durchlaufen werden.

Weiter können noch Tests durchgeführt werden, die die Referenzfunktionen nicht benötigen. Hierbei werden bestimmte mathematische Eigenschaften der implementierten Funktion herangezogen, die beim Entwurf und der Umsetzung nicht verwendet wurden. Ein typisches Beispiel zum Testen von Besselfunktionen (sofern Implementierungen für Besselfunktionen erster und zweiter Art zur Verfügung stehen) ist das Nachprüfen der Wronski-Determinante:

$$J_{n+1}(x)Y_n(x) - J_n(x)Y_{n+1}(x) = \frac{2}{\pi x}.$$

Beim Test von Intervallfunktionen wird anstelle der Gleichheit die Inklusionsbedingung

$$J_{n+1}(x)Y_n(x) - J_n(x)Y_{n+1}(x) \supseteq \frac{2}{\pi x}$$

geprüft werden, für die rechte Seite muß dabei die im verwendeten Gleitkommaformat engstmögliche Einschließung berechnet werden. Alternativ kann

die Bedingung

$$J_{n+1}(x)Y_n(x) - J_n(x)Y_{n+1}(x) \cap \frac{2}{\pi x} \neq \emptyset$$

getestet werden.

Verschiedene weitere Testmöglichkeiten für Gleitkommaimplementierungen von Besselfunktionen werden u.a. von Macleod [87] vorgeschlagen. Einige Grundprinzipien zum Testen von Implementierungen von speziellen Funktionen der mathematischen Physik finden sich auch bei Lozier [82].

Kapitel 6

Zusammenfassung und Ausblick

Ein wesentlicher Teil der vorgestellten Arbeit besteht in der Bereitstellung von mathematischen Grundlagen zur Fehlererfassung und Fehlerabschätzung bei Gleitkommaalgorithmen. Die erarbeiteten Aussagen wurden in effiziente (halb-) automatische Softwarewerkzeuge umgesetzt. Diese Werkzeuge stellen nun eine gute Grundlage für die weitere Entwicklung von Funktionen, die im Rahmen der Verifikationsnumerik eingesetzt werden sollen, zur Verfügung.

Einige dieser Softwarewerkzeuge wurden im Zusammenhang mit der Entwicklung von schnellen elementaren Intervallfunktionen (vgl. Hofschuster/Krämer [51] und [52]) getestet und haben hierbei ihre Mächtigkeit bereits gezeigt.

Verschiedene Ideen und Techniken zur Berechnung von Funktionswerteinschließungen bei speziellen Funktionen der mathematischen Physik wurden exemplarisch anhand der Besselfunktionen untersucht.

Um eine Intervallversion der Besselfunktionen zu erhalten, die in eine existierende Software-Funktionsbibliothek integriert werden kann, können die in dieser Arbeit untersuchten Berechnungsverfahren in Abhängigkeit des Argumentbereichs miteinander kombiniert werden. Bei dieser Softwareumsetzung sollte dann auch eine Optimierung in Bezug auf die Laufzeit (in Abhängigkeit von der gewünschten Rechnerumgebung) vorgenommen werden.

Eine interessante Fragestellung ist, inwieweit sich die vorgestellten Berechnungsmöglichkeiten auch auf Besselfunktionen mit reeller Ordnung oder auf die modifizierten Besselfunktionen erweitern lassen. Interessant wäre auch, die eventuell mögliche Übertragbarkeit der Methoden auf Funktio-

nen, die mit den Besselfunktionen eng verwandt sind, wie z.B. die Airy– oder die Kelvin–Funktionen, zu untersuchen.

Eine weitere Herausforderung ist die Entwicklung entsprechender Funktionsroutinen für komplexe Argumente. Bereits anhand der elementaren Funktionen kann man sehen, daß dies ein nicht immer einfaches Unterfangen ist (vgl. Braune [24] und Krämer [58]).

Ein anzustrebendes Ziel wäre eine einheitliche (Intervall-) Funktionsbibliothek, die die am häufigsten benötigten speziellen Funktionen der mathematischen Physik dem Anwender zur Verfügung stellt.

Anhang A

Grundlagen aus dem Bereich der Intervallrechnung

A.1 Grundlegende Bezeichnungen, Notation

In der vorliegenden Arbeit bezeichnet stets

$\mathbb{R}$	die Menge der reellen Zahlen,
$\mathbb{R}^n$	die Menge der n-dimensionalen Vektoren über $\mathbb{R}$,
$\mathbb{R}^{n\times m}$	die Menge der $n \times m$-Matrizen über $\mathbb{R}$,
$\mathbb{PR}$	die Potenzmenge über den reellen Zahlen,
$\mathbb{PR}^n$	die Potenzmenge über den Vektoren und
$\mathbb{PR}^{n\times m}$	die Potenzmenge über den Matrizen.

Weiter werden folgende Notationen verwendet:

$S = S(b, l, e_{min}, e_{max})$	Gleitkommaraster mit Basis b, Mantissenlänge l und Exponent e mit $e_{min} \leq e \leq e_{max}$
$S(2, 53, -1022, 1023)$	IEEE-Datenformat double (vgl. Abschnitt A.4)
$\circ \in \{+, -, \cdot, /\}$	exakte reelle Operation
$\boxdot, \circ \in \{+, -, \cdot, /\}$	Gleitkommaoperator, Maschinenoperation mit Rundung zur nächstgelegenen Gleitkommazahl
$\bigtriangledown, \circ \in \{+, -, \cdot, /\}$	Maschinenoperation mit Rundung nach unten
$\bigtriangleup, \circ \in \{+, -, \cdot, /\}$	Maschinenoperation mit Rundung nach oben
`MinReal`	kleinste positive normalisierte Gleitkommazahl, für IEEE-Datenformat double gilt:

	`MinReal`:=$2.22\ldots\cdot 10^{-308}$
`MaxReal`	größte normalisierte Gleitkommazahl, für IEEE-Datenformat double gilt: `MaxReal`:=$1.78\ldots\cdot 10^{308}$
$\tilde{a}$	auf der Maschine berechnete, i.a. fehlerbehaftete Größe
ulp	eine Einheit in der letzten Mantissenstelle (unit last place)
ε^*	relative Maschinengenauigkeit, $\varepsilon^* := \frac{1}{2}2^{1-l} = 2^{-l}$
`eps52`	`eps52`:$= 2^{1-53} = 2.22044\ldots\cdot 10^{-16}$
`eps53`	`eps53`:$= \frac{1}{2}2^{1-53} = 1.11022\ldots\cdot 10^{-16} = \varepsilon^*$
$\mathrm{pred}(x),\ x \in S$	Gleitkommavorgänger von x
$\mathrm{succ}(x),\ x \in S$	Gleitkommanachfolger von x
$\mathbb{R}^+$	Menge der positiven reellen Zahlen
$I\mathbb{R}$	Menge der abgeschlossenen Intervalle über den reellen Zahlen
IS	Menge der Maschinenintervalle $IS = \{[\underline{a}, \overline{a}] \mid \underline{a}, \overline{a} \in S, \underline{a} \leq \overline{a}\}$
$x\big\vert_n,\ x \in S,\ n \leq l$	Die führenden n Mantissenbits von x

A.2 Rechnerarithmetik

Da auf einer Rechenanlage nur endlich viele Zahlen darstellbar sind, muß die Menge der reellen Zahlen auf eine Teilmenge, die sogenannten *Gleitkommazahlen* oder auch *Maschinenzahlen*, abgebildet werden. Entsprechendes gilt für die Vektoren und Matrizen über den reellen Zahlen. Außerdem müssen auch die exakten arithmetischen Grundoperationen $+$, $-$, $\cdot$ und $/$ für die inneren und äußeren Verknüpfungen der reellen Räume auf dem Rechner durch die sogenannten *Gleitkommaoperationen* oder auch *Maschinenoperationen* approximiert werden. Kontrolliertes wissenschaftliches Rechnen auf einer Rechenanlage erfordert daher eine mathematisch exakte Definition der Rechnerarithmetik. Diese wurde von Kulisch und Miranker in [71], [72] und [73] angegeben. In diesem Abschnitt wollen wir kurz die wichtigsten Begriffe und Schreibweisen aus diesem Bereich erläutern.

Definition A.2.1 *Unter einem normalisierten Gleitkommasystem* $S = S(b, l, e_{min}, e_{max})$ *versteht man eine Menge reeller, auf einer Rechenanlage darstellbarer Zahlen der Form*

$$x = 0 \text{ oder } x = \pm m \cdot b^e.$$

Hierbei bezeichnet $b \in \mathbb{N}\backslash\{1\}$ *die Basis des Gleitkommasystems,* $l \in \mathbb{N}$ *die Mantissenlänge einer Gleitkommazahl* x, $m = 0.x_1x_2 \ldots x_l$ *mit* $x_1 \neq 0$, $0 \leq x_i < b$, $i = 2(1)l$ *die Mantisse*[1] *und* $e \in [e_{min}, e_{max}] \cap \mathbb{Z}$ *mit* $e_{min}, e_{max} \in \mathbb{Z}$ *den Exponenten.*

Bemerkung:

1. S ist symmetrisch zur Null, d.h. $x \in S \Longrightarrow -x \in S$.

2. Die größte positive (normalisierte) Gleitkommazahl hat den Wert

$$x_{max} := 0.x_1x_2 \ldots x_l \cdot b^{e_{max}} \text{ mit } x_i = b - 1, i = 1(1)l.$$

3. Die kleinste positive (normalisierte) Gleitkommazahl hat den Wert

$$x_{max} := 0.1000 \ldots 0 \cdot b^{e_{min}}.$$

4. Die Menge $\{x \in \mathbb{R} \mid |x| > x_{max}\}$ heißt Überlaufbereich.

5. Die Menge $\{x \in \mathbb{R} \mid |x| \in (0, x_{min})\}$ heißt Unterlaufbereich.

6. Der Unterlaufbereich eines Gleitkommasystems kann mit weiteren, sogenannten denormalisierten Gleitkommazahlen untergliedert werden. Diese Zahlen haben die Form

$$x = \pm 0.\underbrace{000 \ldots 0}_{k-1} x_k \ldots x_l \cdot b^{e_{min}}.$$

7. Die kleinste positive denormalisierte Gleitkommazahl hat den Wert

$$x = \pm 0.\underbrace{000 \ldots 0}_{l-1} 1 \cdot b^{e_{min}}.$$

[1] Teilweise wird für die Mantisse m auch $m = x_1.x_2 \ldots x_l$ verwendet.

Definition A.2.2 *Gegeben sei ein Gleitkommasystem* $S = S(b, l, e_{min}, e_{max})$. *Die Größe*

$$\varepsilon^* := \frac{1}{2} \cdot b^{1-l}$$

bezeichnet man als relative Maschinengenauigkeit.

Im folgenden bezeichnen stets

R	die Menge der Maschinenzahlen,
R^n	die Menge der n-dimensionalen Vektoren über R und
$R^{n\times m}$	die Menge der $n \times m$-Matrizen über R.

Die reellen Räume werden gemäß

$$\begin{array}{lcl} \mathbb{R} & \to & R \\ \mathbb{R}^n & \to & R^n \\ \mathbb{R}^{n\times n} & \to & R^{n\times n} \end{array}$$

auf die Maschinenräume abgebildet. Im folgenden sei S ein Raum der linken Spalte und T der zugehörige Raum der rechten Spalte.

Definition A.2.3 *Die Abbildung* $\square : S \to T$ *mit den Eigenschaften*

(R1) $\square a = a$ *für alle* $a \in T$, *(Projektion)*

(R2) $a \leq b \;\Rightarrow\; \square a \leq \square b$ *für alle* $a, b \in S$, *(Monotonie)*

heißt (monotone) Rundung. Eine Rundung heißt antisymmetrisch, wenn gilt

(R3) $\square(-a) = -\square a$ *für alle* $a \in S$.

Eine Rundung heißt nach unten gerichtet (nach oben gerichtet), wenn gilt

(R4) $\square(a) \leq a$ $(\square(a) \geq a)$ *für alle* $a \in S$.

Alle inneren und äußeren Verknüpfungen $+$, $-$, $\cdot$ und $/$ in S werden durch die Gleitkommaverknüpfungen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ in T approximiert, die über das Prinzip des Semimorphismus definiert sind.

Definition A.2.4 *Eine antisymmetrische Rundung* $\square : S \to T$ *heißt Semimorphismus, wenn alle inneren und äußeren Verknüpfungen in* T *durch*

$$\text{(RG)} \qquad a \boxdot b := \square(a \circ b) \qquad \textit{für alle } a, b \in T \textit{ und } \circ \in \{+, -, \cdot, /\}$$

definiert sind.

Die dadurch erklärten Verknüpfungen für Elemente aus T werden stets zunächst in S ausgeführt und das Ergebnis erst dann wieder nach T gerundet. Semimorphe Verknüpfungen sind damit von *maximaler Genauigkeit*, in dem Sinne, daß zwischen dem in S berechneten Verknüpfungsergebnis $a \circ b$ und seiner Approximation $a \boxdot b$ in T kein weiteres Element aus T liegt (vgl. [71], [72]). Für die Produkträume ist dies komponentenweise zu verstehen.

Beispiel A.2.1 Das Skalarprodukt zweier Gleitkommavektoren $a, b \in R^n$ wird über den Semimorphismus definiert durch

$$a \boxdot b := \square(a \cdot b) = \square\left(\sum_{i=1}^{n} a_i \cdot b_i\right).$$

Definition A.2.5 *Seien $u, v \in S = S(b, l, e_{min}, e_{max})$ beliebig, $\circ \in \{+, -, \cdot, /\}$ und $w := u \circ v$, wobei $v \neq 0$ falls $\circ = /$. Weiter seien x und y zwei aufeinanderfolgende Maschinenzahlen mit $w \in [x, y)$. Eine Gleitkommaarithmetik heißt dann „faithful"[2], wenn gilt:*

$$\begin{aligned} w = x \quad &\Longrightarrow \quad \square(u \circ v) = x \quad \textit{und} \\ w \neq x \quad &\Longrightarrow \quad \square(u \circ v) = x \vee \square(u \circ v) = y. \end{aligned}$$

Bemerkung: Nachfolgend verwenden wir das zuvor als allgemeines Rundungssymbol gebrauchte Zeichen $\square$ für die antisymmetrische Rundung zur nächstgelegenen Maschinenzahl. Die nach unten bzw. nach oben gerichteten Rundungen zur nächstkleineren bzw. nächstgrößeren Maschinenzahl bezeichnen wir mit $\bigtriangledown$ bzw. $\bigtriangleup$. Sie sind eindeutig bestimmt, es gilt die Identität

$$\bigtriangledown(-x) = - \bigtriangleup x \quad \text{für alle } x \in S.$$

Wir verwenden die Bezeichnung $f_{\square}(x)$ für die Maschinenauswertung einer Funktion f, d. h. einer Berechnung des Funktionswertes unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$. Den Begriff maximale Genauigkeit verwenden wir auch im Zusammenhang mit Maschinenauswertungen $s_{\square}$ von Funktionen $s : I\!R \to I\!R$, die durch

$$s_{\square}(x) := \square(s(x))$$

[2] vgl. Dekker [35], Linnainmaa [76]

ebenfalls über den Semimorphismus definiert sind. Den auf der Rechenanlage zur Verfügung stehenden Satz maximal genauer elementarer Funktionen $s : R \to R$ bezeichnen wir im folgenden mit

$$\mathcal{SF} = \{\exp, \ln, \sin, \cos, \ldots\}.$$

A.3 Intervallarithmetik, Maschinenintervallarithmetik

Das wesentliche Hilfsmittel zur Berechnung von Schranken für die Lösung eines numerischen Problems und damit zur Gewinnung von garantierten Aussagen ist die Intervallrechnung. Eine ausführliche Definition und Darstellung der Intervallrechnung findet sich bei Alefeld/Herzberger [2], [3] oder Moore [93], eine gelungene Einführung z. B. in Mayer [90]. Die Behandlung der zugehörigen Maschinenintervallarithmetik findet sich in Kulisch [71] und Kulisch/Miranker [72]. Im folgenden geben wir eine Zusammenstellung der wichtigsten Begriffe und Eigenschaften.

Definition A.3.1 *Die Menge* $A := [\underline{A}, \overline{A}] := \{x \in I\!R \mid \underline{A} \leq x \leq \overline{A}\}$ *mit* $\underline{A}, \overline{A} \in I\!R$ *heißt Intervall.* $\underline{A} = \inf A$ *heißt Infimum oder Unterschranke von* A, $\overline{A} = \sup A$ *heißt Supremum oder Oberschranke von* A. *Ein Intervall* A *heißt Punktintervall, wenn gilt* $\underline{A} = \overline{A}$.

Es bezeichnet im folgenden stets

$I\!I\!R$	die Menge der Intervalle,
$I\!I\!R^n$	die Menge der n-dimensionalen Vektoren über $I\!I\!R$ und
$I\!I\!R^{n\times m}$	die Menge der $n \times m$-Matrizen über $I\!I\!R$.

Für $A \in I\!I\!R$, $X \in I\!I\!R^n$ und $M \in I\!I\!R^{n\times n}$ verwenden wir die Bezeichnungen

$$\begin{aligned}
m(A) &:= \frac{1}{2}(\underline{A} + \overline{A}) && \text{(Mittelpunkt)},\\
d(A) &:= \overline{A} - \underline{A} && \text{(Durchmesser)},\\
d(X) &:= \max_{1\le i\le n} d(X_i) && \text{(Durchmesser(maximum))},\\
|A| &:= \max\{|a| \mid a \in A\} && \text{(Betrag)},\\
\langle A\rangle &:= \min\{|a| \mid a \in A\} && \text{(Betragsminimum)},\\
\|x\|_\infty &:= \max_i |x_i| && \text{(Maximumnorm) und}\\
\|M\|_\infty &:= \max_i \sum_{j=1}^{n} |M_{ij}| && \text{(Zeilensummennorm)}.
\end{aligned}$$

Dabei sind Mittelpunkt und Betrag für Vektoren und Matrizen jeweils komponentenweise definiert. Für $A, B \in I\!I\!R$ gelten die Beziehungen

$$\begin{aligned}
d(A \pm B) &= d(A) + d(B),\\
d(A \cdot B) &\le d(A)\cdot|B| + d(B)\cdot|A|,\\
|A + B| &\le |A| + |B|,\\
|A \cdot B| &= |A|\cdot|B|.
\end{aligned}$$

Die Beweise finden sich z. B. in Alefeld/Herzberger [2]. Als weitere Schreibweisen für $A \in I\!I\!R$ und $X \in I\!I\!R^n$ benötigen wir

$$\begin{aligned}
\overset{\circ}{A} &:= \{a \in A \mid \underline{A} < a < \overline{A}\} && \text{(Inneres von } A),\\
\overset{\circ}{X} &:= \{x \in X \mid \underline{X_i} < x_i < \overline{X_i} \text{ für alle } i\} && \text{(Inneres von } X),\\
\partial A &:= \{\underline{A}, \overline{A}\} && \text{(Rand von } A),\\
\partial X &:= \{x \in X \mid x_i \in \partial X_i \text{ für ein } i\} && \text{(Rand von } X).
\end{aligned}$$

Vergleiche und Teilmengenbeziehungen werden mengentheoretisch interpretiert und sind für Vektoren und Matrizen genau dann erfüllt, wenn sie für alle ihre Komponenten erfüllt sind. Für $A, B \in I\!I\!R$ gilt

$$\begin{aligned}
A = B &\iff \underline{A} = \underline{B} \ \wedge\ \overline{A} = \overline{B},\\
A \subseteq B &\iff \underline{A} \ge \underline{B} \ \wedge\ \overline{A} \le \overline{B},\\
A \overset{\circ}{\subset} B &\iff \underline{A} > \underline{B} \ \wedge\ \overline{A} < \overline{B} \iff A \subseteq \overset{\circ}{B}\,.
\end{aligned}$$

Die Verbandsoperationen $\cap$ und $\cup$ für $A, B \in I\!I\!R$ sind definiert durch

$$\begin{aligned} A \cap B &:= [\max\{\underline{A}, \underline{B}\}, \min\{\overline{A}, \overline{B}\}], \qquad \text{(Schnitt)} \\ A \cup B &:= [\min\{\underline{A}, \underline{B}\}, \max\{\overline{A}, \overline{B}\}], \qquad \text{(Intervallhülle)} \end{aligned}$$

wobei $A \cap B$ nur definiert ist, falls $\max\{\underline{A}, \underline{B}\} \leq \min\{\overline{A}, \overline{B}\}$ erfüllt ist. Die intervallarithmetischen Operationen werden definiert durch

$$A \circ B := \{a \circ b \mid a \in A,\, b \in B\},$$

wobei $A, B \in I\!I\!R$ und $\circ \in \{+, -, \cdot, /\}$. Die explizite Berechnung dieser Intervalloperationen kann durch

$$\begin{aligned} A \;+\; B &= [\underline{A} + \underline{B},\ \overline{A} + \overline{B}], \\ A \;-\; B &= [\underline{A} - \overline{B},\ \overline{A} - \underline{B}], \\ A \;\cdot\; B &= [\min\{\underline{A}\,\underline{B}, \underline{A}\overline{B}, \overline{A}\underline{B}, \overline{A}\,\overline{B}\},\ \max\{\underline{A}\,\underline{B}, \underline{A}\overline{B}, \overline{A}\underline{B}, \overline{A}\,\overline{B}\}], \\ A \;/\; B &= [\underline{A}, \overline{A}] \cdot [1/\overline{B},\ 1/\underline{B}] \quad \text{für } 0 \notin B \end{aligned}$$

erfolgen. Dabei kann die Bildung des Minimums bzw. des Maximums in der Multiplikation durch vorherige Untersuchung der Intervallgrenzen in den meisten Fällen vermieden werden. Im Hinblick auf die in Abschnitt 1.3 beschriebenen Erweiterungen geben wir noch eine entsprechende Formulierung mit Fallunterscheidung für die Division A/B, falls $0 \notin B$:

$$A/B = \begin{cases} [\overline{A}/\underline{B}, \underline{A}/\overline{B}] & \text{für } \overline{A} \leq 0 \ \wedge\ \overline{B} < 0 \\ [\underline{A}/\underline{B}, \overline{A}/\overline{B}] & \text{für } \overline{A} \leq 0 \ \wedge\ 0 < \underline{B} \\ [\overline{A}/\overline{B}, \underline{A}/\overline{B}] & \text{für } \underline{A} < 0 < \overline{A} \ \wedge\ \overline{B} < 0 \\ [\underline{A}/\underline{B}, \overline{A}/\underline{B}] & \text{für } \underline{A} < 0 < \overline{A} \ \wedge\ 0 < \underline{B} \\ [\overline{A}/\overline{B}, \underline{A}/\underline{B}] & \text{für } 0 \leq \underline{A} \ \wedge\ \overline{B} < 0 \\ [\underline{A}/\overline{B}, \overline{A}/\underline{B}] & \text{für } 0 \leq \underline{A} \ \wedge\ 0 < \underline{B} \end{cases} \tag{A.1}$$

Addition und Multiplikation sind kommutativ und assoziativ, es gilt jedoch nur die sogenannte *Subdistributivität*

$$A \cdot (B + C) \subseteq A \cdot B + A \cdot C$$

für Intervalle $A, B, C \in I\!I\!R$. Eine weitere, zentrale Eigenschaft der Intervalloperationen ist die *Inklusionsisotonie*

$$\begin{aligned} a \in A \ \wedge\ b \in B &\implies a \circ b \in A \circ B \\ A \subseteq C \ \wedge\ B \subseteq D &\implies A \circ B \subseteq C \circ D \end{aligned}$$

für alle $\circ \in \{+, -, \cdot, /\}$ mit $a, b \in I\!R$ und $A, B, C, D \in I\!I\!R$.

Mit Hilfe der Intervallarithmetik ist es möglich, den Wertebereich einer Funktion einzuschließen. In diesem Zusammenhang verwenden wir für Funktionen $f : D \to I\!R$ mit $D \subseteq I\!R$ oder $D \subseteq I\!R^n$ die Bezeichnungen

- $f(X)$ für den *Wertebereich* von f,
- $F(X)$ für die *Intervallauswertung*, also für die Auswertung der *Intervallerweiterung* oder auch *Einschließungsfunktion* F von f,
- $\overline{F_X}$ für $\sup F(X)$ und
- $\underline{F_X}$ für $\inf F(X)$,

wobei für $f(X)$ und $F(X)$ stets die Beziehung

$$f(X) \subseteq F(X)$$

gilt. Entsprechende Bezeichnungen gelten auch für vektorwertige Funktionen. Zur *natürlichen Intervallerweiterung* eines Ausdrucks f kommt man, indem man alle auftretenden Variablen durch entsprechende Intervalle und alle Operationen durch die zugehörigen Intervalloperationen ersetzt. In Teilausdrücken können dabei auch sogenannte elementare Intervallfunktionen $s \in \mathcal{SF}$ vorkommen, für die gilt

$$s(X) = S(X).$$

Für Punktintervalle bzw. Punktintervallvektoren als Argumente von f gilt stets

$$f(c) = F(c).$$

Auch für die Intervallerweiterungen gilt die Inklusionsisotonie

$$\begin{aligned} a \in A &\implies f(a) \in F(A) \\ A \subseteq B &\implies F(A) \subseteq F(B). \end{aligned}$$

Aufgrund der Tatsache, daß die natürliche Intervallauswertung üblicherweise große Überschätzungen mit sich bringt, werden häufig auch *zentrierte Formen* oder *Mittelwertformen* verwendet (vgl. auch Ratschek/Rokne [102]). Die Mittelwertform wird aus dem Mittelwertsatz abgeleitet und ist für $f : D \subseteq I\!R \to I\!R$ durch

$$F^c(X) = f(c) + F'(X) \cdot (X - c)$$

und für $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ durch

$$F^c(X) = f(c) + \nabla F(X) \cdot (X - c)$$

definiert, wobei $c \in X$ ist und meistens $c = m(X)$ verwendet wird.

Beim Rechnen mit Intervallen auf einer Rechenanlage muß die Menge der reellen Intervalle auf die Menge der *Maschinenintervalle* abgebildet werden. Dazu werden die Intervallgrenzen durch gerichtete Rundungen auf Maschinenzahlen abgebildet. Man beachte jedoch, daß ein Intervall auch nach diesem Übergang auf ein Maschinenintervall

$$A := [\underline{A}, \overline{A}] := \{x \in \mathbb{R} \mid \underline{A} \leq x \leq \overline{A};\ \underline{A}, \overline{A} \in R\}$$

sämtliche *reelle* Werte zwischen den Grenzen repräsentiert, also nach wie vor das gesamte Kontinuum abdeckt. Entsprechendes gilt für die Vektoren und Matrizen über den reellen Intervallen. Somit müssen auch die exakten arithmetischen Grundoperationen $+$, $-$, $\cdot$ und $/$ für die inneren und äußeren Verknüpfungen der reellen Intervallräume auf dem Rechner durch die sogenannten *Maschinenintervalloperationen* approximiert werden.
Es bezeichnen stets

IR	die Menge der Maschinenintervalle,
IR^n	die Menge der n-dimensionalen Vektoren über IR und
$IR^{n\times m}$	die Menge der $n \times m$-Matrizen über IR.

Die reellen Intervallräume werden gemäß

$$\begin{array}{lcl} I\mathbb{R} & \to & IR \\ I\mathbb{R}^n & \to & IR^n \\ I\mathbb{R}^{n\times n} & \to & IR^{n\times n} \end{array}$$

auf die Maschinenräume abgebildet. Im folgenden sei IS ein Raum der linken Spalte und IT der zugehörige Raum der rechten Spalte.

Definition A.3.2 *Die Abbildung $\diamondsuit : IS \to IT$ mit den Eigenschaften*

(R1) $\diamondsuit A = A$ *für alle* $A \in IT$,

(R2) $A \subseteq B \ \Rightarrow\ \diamondsuit A \subseteq \diamondsuit B$ *für alle* $A, B \in IS$,

(R3) $\diamondsuit(-A) = -\diamondsuit A$ *für alle* $A \in IS$,

(R4) $A \subseteq \diamondsuit A$ *für alle* $A \in IS$,

heißt antisymmetrische, nach außen gerichtete Rundung oder auch Intervallrundung.

Bemerkung: Die Rundung $\diamondsuit$ ist eindeutig bestimmt (vgl. Kulisch [71], Kulisch/Miranker [72]).

Alle inneren und äußeren Verknüpfungen $+$, $-$, $\cdot$ und $/$ in IS werden durch die Maschinenintervallverknüpfungen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$ in IT approximiert, die über das Prinzip des Semimorphismus durch

$$\text{(RG)} \qquad A \diamondsuit B := \diamondsuit(A \circ B) \qquad \text{für alle } A, B \in IT \text{ und } \circ \in \{+, -, \cdot, /\}$$

definiert sind. Wir verwenden die Bezeichnung $F_\diamondsuit(X)$ für die Maschinenintervallauswertung einer Intervallfunktion F unter Verwendung der gerundeten Operationen $\boxplus$, $\boxminus$, $\boxdot$ und $\boxslash$. Maschinenintervallfunktionen $S_\diamondsuit$ der Intervallfunktionen $S : \mathbb{IR} \to \mathbb{IR}$ mit $S \in \mathcal{SF}$ (Standardfunktionen) sind über den Semimorphismus definiert durch

$$S_\diamondsuit(X) := \diamondsuit(S(X)).$$

Die Inklusionsisotonie gilt für die Maschinenenoperationen in der Form

$$\begin{aligned} a \in A \ \wedge \ b \in B \quad &\Longrightarrow \quad a \circ b \in A \circ B \subseteq A \diamondsuit B \\ A \subseteq C \ \wedge \ B \subseteq D \quad &\Longrightarrow \quad A \diamondsuit B \subseteq C \diamondsuit D \end{aligned}$$

für alle $\circ \in \{+, -, \cdot, /\}$ mit $a, b \in R$ und $A, B, C, D \in I\,R$ und für die Maschinenintervallauswertungen in der Form

$$\begin{aligned} a \in A \quad &\Longrightarrow \quad f(a) \in F(A) \subseteq F_\diamondsuit(A) \\ A \subseteq B \quad &\Longrightarrow \quad F_\diamondsuit(A) \subseteq F_\diamondsuit(B). \end{aligned}$$

Mittelpunkt und Durchmesser werden auf dem Rechner als

$$m_\square(X) = \square(m(X)) \quad \text{und} \quad d_\triangle(X) = \triangle(d(X))$$

definiert, wobei zu beachten ist, daß m und d damit zwar maximal genau sind, daß jedoch im allgemeinen gilt

$$m_\square(X) \neq m(X) \quad \text{bzw.} \quad d_\triangle(X) \neq d(X).$$

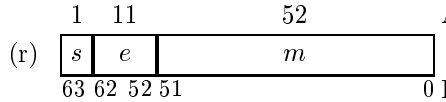
A.4 IEEE–Standard 754

Im IEEE-Standard 754 [10], der im Jahre 1985 veröffentlicht wurde, werden vier Datenformate für Gleitkommazahlen (single, single extended, double und double extended) und deren Kodierung auf der Maschine definiert. Weiter werden vier verschiedene Rundungsarten (round to nearest, directed rouning toward $-\infty$, 0, $+\infty$) zur Rundung einer Zahl in eines der vier Datenformate vorgegeben und es werden Operationen (Addition, Subtraktion, Multiplikation, Division, Wurzelfunktion, Restbildung, Ungleichungen), Konvertierungen zwischen den einzelnen Datenformaten und die Behandlung bestimmter Größen ($\pm\infty$, ± 0, NaN) standardisiert. Regeln zur Ausnahmebehandlung im Fehlerfall schließen den Standard ab.

Das im IEEE–Standard beschriebene „double"–Format mit dem Gleitkommasystem $S(B, l, e_{min}, e_{max}) = S(2, 53, -1022, 1023)$ wird im folgenden zugrunde gelegt. Hierbei handelt es sich um ein binäres Zahlenformat mit einer Länge von 64 Bit (8 Byte). Der Unterlaufbereich wird mit denormalisierten Gleitkommazahlen aufgefüllt (vgl. Bemerkung A.2). Zur Darstellung der Mantisse werden dabei 53 Bit, des Exponenten 11 Bit und für das Vorzeichen 1 Bit verwendet. Bei der Speicherung der Mantisse wird die sogenannte Technik des „Hidden Bit" verwendet. Da die Mantisse einer normalisierten Zahl immer mit der Ziffer 1 und die Mantisse einer denormalisierten Zahl immer mit der Ziffer 0 beginnt, braucht dieses führende Bit nicht abgespeichert zu werden, zur Speicherung der Mantisse genügen also 52 Bit. Um eine Vorzeichenbetrachtung des Exponenten zu vermeiden, wird diesem vor dem Speichern die Konstante bias:= 1023 hinzuaddiert, d.h. die Darstellung erfolgt immer als positive (bzw. vorzeichenlose) Zahl. Für die Werte $+\infty$ und $-\infty$, sowie für NaNs (Not a Number)[3] sind spezielle Kodierungen vorgesehen.

Eine Bit–Darstellung[4] einer Zahl des „double"–Formates kann Abbildung A.1 entnommen werden.

[3] Im Standard werden mindestens ein „signaling NaN" und ein „quiet NaN" gefordert.
[4] Auf der Maschine dürfen die einzelnen Bits auch in einer anderen Reihenfolge gespeichert werden.



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

Abbildung A.1: IEEE double–Zahlformat

Die größte und die kleinste darstellbare positive Zahl wird mit `maxreal` bzw. `minreal` bezeichnet (Tabelle A.1).

Hexadezimale Darstellung	Dezimaler Wert
`0000000000000001` `minreal`	$4.9406564584124654 \cdot 10^{-324}$
`7FEFFFFFFFFFFFFF` `maxreal`	$1.7976931348623158 \cdot 10^{308}$

Tabelle A.1: Werte für `minreal` und `maxreal` im IEEE double–Format

Hexadezimale Darstellung	Rundungsmodus	Dezimaler Wert
`3FF0000000000000`	exakt	1.0
`BFF0000000000000`	exakt	-1.0
`3FE0000000000000`	exakt	0.5
`3FB999999999999A`	$\square$	0.1
`3FB9999999999999`	∇	0.1
`3FB999999999999A`	$\triangle$	0.1
`400921FB54442D18`	$\square$	π
`3FF921FB54442D18`	$\square$	$\pi/2$

Tabelle A.2: Darstellung einiger wichtiger Dezimalzahlen im IEEE double–Format

Die speziell kodierten Werte „Not a Number“ (NaN) werden für die Aus-

nahmebehandlung im Fehlerfalle verwendet.
Die Darstellung einer spezieller Werte im IEEE double–Format werden in der Tabelle A.3 angegeben.

Hexadezimale Darstellung	Dezimalzahl
FFF0000000000000	$-\infty$
FFEFFFFFFFFFFFFF	$-1.7976931348623158 \cdot 10^{308}$
BFF0000000000000	-1.0
8010000000000000	$-2.2250738585072013 \cdot 10^{-308}$
8000000000000001	$-4.9406564584124654 \cdot 10^{-324}$
8000000000000000	-0.0
0000000000000000	0.0
0000000000000001 minreal	$4.9406564584124654 \cdot 10^{-324}$
0010000000000000	$2.2250738585072013 \cdot 10^{-308}$
3FF0000000000000	1.0
7FEFFFFFFFFFFFFF maxreal	$1.7976931348623158 \cdot 10^{308}$
7FF0000000000000	$+\infty$

Tabelle A.3: Einige wichtige Grössen im IEEE–double–Format

Bemerkung: Eine Gleitkommaarithmetik, die dem IEEE–Standard 754 genügt, ist im Sinne von Definition A.2.5 im Bereich der normalisierten Zahlen „faithful“.

A.5 XSC–Sprachen

Die Programmiersprachen Pascal–XSC [55, 56] und C–XSC [57] sind Erweiterungen der Sprachen Pascal bzw. C++. Sie wurden mit dem Ziel entwickelt, Werkzeuge für die numerische Lösung wissenschaftlicher Probleme zur Verfügung zu stellen. Das hierbei zur Verfügung gestellte Operatorkonzept, das Modulkozept (in Pascal–XSC) und die Möglichkeit, Operationen mit gerichter Rundung anzusprechen, wird in dieser Arbeit für die Implementierung des Fehlerkalküls ausgiebig genutzt.

Anhang B

Quellcode einiger Programme

B.1 Fehlerkalkül

Im folgenden werden einige Ausschnitte aus der Implementierung des Pascal–XSC Moduls `abs_ari` (absoluter Fehlerkalkül) abgedruckt.

Definition des Datentyps:

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

Hilfsfunktion zur Erkennung der Anzahl der Mantissenbits mit dem Wert 0 am Ende der Mantisse einer Gleitkommazahl: der

```
type twoint = record
                case boolean of
                  false : ( a     : array[1..8] of char ) ;
                  true  : ( r     : real    ) ;
              end ;

function tz_test(x:real):integer;
var help:twoint;
    anz,i,k:integer;
begin
  help.r:=x;
```

```
{ for i:=1 to 8 do write(ord(help.a[i]),' '); writeln; }

  anz:=0;
  if intel then begin
     i:=1;
     while ((ord(help.a[i])=0) and (i<=6)) do begin
       anz:=anz+8;
       i:=i+1;
     end;

     if (i=7) then begin
       k:=2;
       while (( ord(help.a[i]) mod k = 0 ) and ( k<=16 )) do
       begin
          anz:=anz+1;
          k:=k*2;
       end;
     end else begin
       k:=2;
       while (( ord(help.a[i]) mod k = 0 ) and ( k<=256 )) do
       begin
          anz:=anz+1;
          k:=k*2;
       end;
     end;

  end else begin
     i:=8;
     while ((ord(help.a[i])=0) and (i>=3)) do begin
       anz:=anz+8;
       i:=i-1;
     end;

     if (i=2) then begin
       k:=2;
       while (( ord(help.a[i]) mod k = 0 ) and ( k<=16 )) do
       begin
          anz:=anz+1;
          k:=k*2;
       end;
     end else begin
       k:=2;
       while (( ord(help.a[i]) mod k = 0 ) and ( k<=256 )) do
```

```
      begin
         anz:=anz+1;
         k:=k*2;
      end;
    end;

  end;
  tz_test:=anz;
end; { tz_test }
```

Operator zur Addition zweier Zahlen:

```
global function DeltaAdd(
               alpha, beta: interval; DeltaA, DeltaB: real ): real;
  {-------------------------------------------------------------}
  { Berechnet wird die absolute Fehlerschranke bei einer        }
  {   Addition von fehlerbehafteten Groessen.                   }
  {-------------------------------------------------------------}
  { Die exakten Werte des 1. Summanden liegen im Intervall alpha.}
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaA beschraenkt.                                         }
  { Die exakten Werte des 2. Summanden liegen im Intervall beta. }
  { Der absolute Fehler der fehlerbehafteten Werte ist durch    }
  { DeltaB beschraenkt.                                         }
  {-------------------------------------------------------------}
var u, v: real;
    ResultSet: interval;
begin
  if (DeltaA=0) and (DeltaB=0) and ((alpha=0) or (beta=0)) then
    DeltaAdd:= 0
  else begin
    if EpsAriTest then begin  { Unterlaufwarnung }
      ResultSet:= alpha + intval(-DeltaA, DeltaA)
                        + beta + intval(-DeltaB, DeltaB);
      if not (ResultSet >< UnflowRange) then begin
        write('  DeltaAdd: Ergebnis im Unterlauf!
                                          Weiter mit <Return> ');
        readln;
      end;
    end; { Unterlaufwarnung }
    u:= EpsQuer *> MaxAbs(alpha+beta);
    v:= (DeltaA +> DeltaB) *> (1 +> EpsQuer);
    DeltaAdd:= u +> v +> MinReal;
  end;
```

```
end;


global operator + (x, y: BoundType) erg: BoundType;
begin
  erg.AbsErr    := DeltaAdd(x.Enclosure, y.Enclosure,
                                          x.AbsErr, y.AbsErr);
  erg.Enclosure := x.Enclosure + y.Enclosure;
end;
```

B.2 Approximationsfehlerbestimmung

Nachfolgend wird der Programmquellcode zur Einschließung des Approximationsfehlers für die in Abschnitt 3.3, Seite 70 angegebene Approximation von Hart ('JZERO 5847') abgedruckt.

```
{-------------------------------------------------------------}
{  Verifikation der Fehlerkurve einer rationalen Approximation  }
{-------------------------------------------------------------}
program J0g; { Approximation lt. Hart et. al. JZER05847 }

use i_ari;       { Intervallarithmetik                    }
use mp_ari;      { Reelle Langzahlrechnung                }
use mpi_ari;     { Langzahlintervallrechnung              }
use x_real;      { Hexadezimale Ausgabe, ...              }

const MaxDegree = 200;  { Maximaler Grad bei Produktpolynomen       }
const np = 10;      { Zaehlergrad der rationalen Approximation      }
const nq =  4;      { Nennergrad der rationalen Approximation       }
const ns = 25;      { Grad der nahezu perfekten Reihenapproximation }
const alpha= 1e-22; { Abbruchfehlerschranke im betrachteten Bereich }
const x0 = 0.0;     { Entwicklungsstelle                            }
const Prec = 6;     { Genauigkeit der Langzahlintervallarithmetik   }

type poly    = array[0..MaxDegree] of real;
type ipoly   = array[0..MaxDegree] of interval;
type mpipoly = array[0..MaxDegree] of mpinterval;
var p, q:  poly; { Zaehler- u. Nennerpolynom der rat. Approximation }
    s:  mpipoly; { Nahezu perfekte Approximation                    }
    Diff: ipoly; { Differenzpolynom s(x)*q(x)-p(x)                  }
    fPolEncl: array [0.. ns] of interval; { Einschliessung von f    }
```

```
    RelErr: boolean;
    Prot: text;      { Datei zum Protokollieren der Berechnungen }
    iPol: text;      { Koeffizienten des Fehlerpolynoms          }

function FACtoRIAL(k: integer) : mpinterval;
{*************************************************************}
{*                                                           *}
{*  Berechnet eine  garantierte  Einschliessung der Fakultaet   *}
{*       k! = 1*2*3*...*k;     k = 1,2,...,270;     0! = 1;     *}
{*                                                           *}
{*************************************************************}
var p : integer;
    t : mpinterval;

begin
  FACtoRIAL := TRUE;  mpinit(t);
  case k of
  0,1 : t := 1 ;
  else: begin
          t := 1;
          For p := 2 to k do t := t * _mpinterval(p);
        end;
  end; { case }
  FACtoRIAL := t;
  mpfree(t);
  FACtoRIAL := FALSE;
end; { FACtoRIAL }

procedure ComputeDiff(s: mpipoly; q, p: poly; var Diff: ipoly);
{ Bestimme Intervallkoeffizienten des Differenzpolynoms }
{   Diff(x) = s(x)*q(x)-p(x)                              }
var h: mpinterval;  { Hilfsgroesse fuer mehrfach genaue Rechnung }
    k, j: integer;
    Koeff: interval;
begin
  mpinit(h);  { Initialisierung der mp-Variablen }
  { Grad des Ergebnispolynoms: Grad s + Grad q,
                                       d.h. Grad Diff:= ns+nq }
  writeln(iPol, ns+nq);
  for k:= 0 to ns+nq do begin
    h:= 0;
    for j:= 0 to k do begin
       h:= h + s[j]*q[k-j];
```

```
    end;
    h:= h - p[k];
    Koeff:= h; { Mehrfachgenaues Intervall in 'normales' Intervall }
    Diff[k]:= Koeff;   { Einschluss des k-ten Polynomkoeffizienten }
    writeln(iPol, k, Koeff);
    writeln(Prot, 'Diff(', k:3, ')= ', Diff[k]);
    if diam(Koeff)=0 then begin
       writeln('Koeffizient Diff(', k:0, ') ist Punktintervall!');
       writeln(Prot, 'Koeffizient Diff(', k:0, ') ist Punktintervall!');
    end
    else if succ(inf(Koeff)) < sup(Koeff) then begin
       writeln('Koeffizient Diff(', k:0, ') ist nicht maximal genau!');
       writeln(Prot, 'Koeffizient Diff(', k:0,
                                       ') ist nicht maximal genau!');
    end;
  end;
  mpfree(h);  { Freigabe der lokalen mp-Variablen }
end;

{ Auswertung des Nennerpolynoms q(x) }
function EvalQ(x: interval): interval;
var k: integer;
    s: interval;
begin
  s:= intval(q[nq]);
  for k:= nq-1 downto 0 do
    s:= s*(x-x0) + intval(q[k]);
  EvalQ:= s;
end;

{ Auswertung des Differenzpolynoms Diff(x) }
function EvalDiff(x: interval): interval;
var k: integer;
    s: interval;
begin
  s:= diff[nq+ns];
  for k:= nq+ns-1 downto 0 do
    s:= s*(x-x0) + Diff[k];
  EvalDiff:= s;
end;

{ Fehlerkurve der Approximation }
function Err(x: interval): interval;
```

```
{ Die Fehlerkurve wird durch               }
{            ( s(x)*q(x) - p(x) ) / q(x)   }
{  beschrieben.                            }
begin
  Err:= EvalDiff(x) / EvalQ(x) + intval(-alpha, alpha);
end;

{ Einschliessung des Wertebereiches von f ueber dem Intervall x }
{ Es wird die nahezu perfekte Approximation s verwendet.        }
function fEncl(x: interval): interval;
var k: integer;
    s: interval;
begin
  s:= fPolEncl[ns];
  for k:= ns-1 downto 0 do
    s:= s*(x-x0) + fPolEncl[k];
  fEncl:= s + intval(-alpha, alpha);  { Fehler beruecksichtigen }
end;

{-------------------------------------------------------------}
var ProgName: string;
             { Programmname zur Kennzeichnung von Ausgabedateien }
    k, n: integer;
    h: real;
    PlotScale: real; { Scaling for GNUPLOT                      }
    Anz, AnzTeilInt, TeilIntProPixel: integer;
    c: mpinterval;
    X: interval;    { Aktuelles Fenster                         }
    ErrX: interval; { Err(X): Auswertung ueber aktuellem Fenster }
    out1, out2, out3, outall: text; { Daten fuer GnuPlot erzeugen}
    InfErrX, SupErrX, MidErrX: real;
    Left, Right: real;   { Grenzen des Approximationsintervalls  }
    Mini, MiniUb: real;  { Einschluss des minimalen Fehlers      }
    MiniX: interval;     { Hier tritt der minimale Fehler auf    }
    Maxi, MaxiLb: real;  { Einschluss des maximalen Fehlers      }
    MaxiX: interval;     { Hier tritt der maximale Fehler auf    }
    { max( |Mini|,|Maxi| ) ist die gesuchte sichere Oberschranke }
    {   von |f - p/q|  bzw von |(f -p/g)/f| im betrachteten      }
    {   Approximationsintervall [Left, Right].                  }
begin
  ProgName:='jzero_5847';
  rewrite(Prot,   ProgName+ '.out');
                         { Protokoll der Berechnungen           }
```

```
  rewrite(out1,   ProgName+ '1.gnu');
                        { Zum Plotten von inf(Err) mit Gnuplot }
  rewrite(out2,   ProgName+ '2.gnu');
                        { Zum Plotten von mid(Err)             }
  rewrite(out3,   ProgName+ '3.gnu');
                        { Zum Plotten von sup(Err)             }
  rewrite(outall, ProgName+ '.gnu');
                        { Zum Plotten aller Kurven mit Gnuplot }
  rewrite(iPol  , ProgName+ 'koeff');
                        { Koeffizienten von err(x)             }

  writeln(Prot, 'Ausgabe des Programms jzero_5847.p');
  writeln(Prot, 'Die Variable ProgName hat den Wert: ', ProgName);
  writeln(Prot);
  writeln(Prot, 'Approximationsfehler einer rat. Approximation fuer ');
  writeln(Prot, '   f(x) = J0(x))    ( err(x):= J0(x)-p(x)/q(x) ) ');
  writeln(Prot);
  setprec(Prec);   { Genauigkeit der Langzahlintervalloperationen }
  writeln(Prot, 'Genauigkeitseinstellung: setprec(', Prec:0, ')');

  { Taylor Reihe fuer f(x) = J0(x))                              }
  { Berechne Intervallkoeffizienten der                          }
  {   nahezu perfekten Approximation an f(x).                    }
  for k:= 0 to MaxDegree do begin
    mpinit(s[k]); { Initialisiere Langzahlintervallkoeffizienten }
    s[k]:=0;
  end;

  writeln(      'Koeffizienten der nahezu perfekten Approximation:');
  writeln(Prot, 'Koeffizienten der nahezu perfekten Approximation:');
  k:= 0;
  s[k]:= 1;
  fPolEncl[k]:= s[k];
  writeln(      k:3, ' ', fPolEncl[k]);
  writeln(Prot, k:3, ' ', fPolEncl[k]);
  k:= 1;
  s[k]:= -0.25;
  fPolEncl[k]:= s[k];
  writeln(      k:3, ' ', fPolEncl[k]);
  writeln(Prot, k:3, ' ', fPolEncl[k]);
  for k:= 2 to ns do begin
    if odd(k) then
       s[k]:=  -1/( POWER(2,2*k)*factorial(k)*factorial(k) )
```

```
  else
     s[k]:=   1/( POWER(2,2*k)*factorial(k)*factorial(k) );
  fPolEncl[k]:= s[k];  { Einschluss durch 'normales' Intervall }
  writeln(k:3, ' ', fPolEncl[k]);
  writeln(Prot, k:3, ' ', fPolEncl[k]);
end;

{ Es wird jeweils der angegebene Dezimalwert zur       }
{ naechstgelegenen Gleitkommazahl gerundet.            }
 p[0] :=  0.588286746328683414238359609e+11;
 p[1] := -0.143437011647514736949175549 8e+11;
 p[2] :=  0.829327521375254047689948 1164e+9;
 p[3] := -0.201017233941079413216720 4749e+8;
 p[4] :=  0.256405434252440745566794 2213e+6;
 p[5] := -0.193446793600678130036000 3678e+4;
 p[6] :=  0.919317910665853136393773 5489e+1;
 p[7] := -0.282517418148632456327500 7924e-1;
 p[8] :=  0.555291373160711194929939 9124e-4;
 p[9] := -0.649436708930119365697570 1941e-7;
 p[10]:=  0.353077221704560439178112 3086e-10;

 writeln(Prot, 'Polynomkoeffizienten des Zaehlerpolynoms: ');
 for k:= 0 to np do
    writeln(Prot, p[k]:'X', '   ', p[k]);

{ Es wird jeweils der angegebene Dezimalwert zur       }
{ naechstgelegenen Gleitkommazahl gerundet.            }
q[0]:= 0.588286746328683429346629 9376e+11;
q[1]:= 0.363467493465600874106423 7087e+9;
q[2]:= 0.996353603100060267502727 7824e+6;
q[3]:= 0.146434177625559953978943 5142e+4;
q[4]:= 0.1e+1;

writeln(Prot, 'Polynomkoeffizienten des Nennerpolynoms: ');
for k:= 0 to nq do
   writeln(Prot, q[k]:'X', '   ', q[k]);

ComputeDiff(s, q, p, Diff);
                { Polynom Diff(x)  =  s(x)*q(x) - p(x) }
                { Der Grad von Diff(x) ist ns+nq.      }

RelErr:= false; { Es wird der Maximale absolute Fehler gesucht! }
writeln(Prot, 'RelErr = ', RelErr);
```

```
{ Betrachtetes Approximationsintervall = [0, 64] }
Left:=  0;
Right:= 64;
PlotScale:= 1E16;
writeln(Prot, 'Fehlerschranke alpha: ', alpha);
writeln(Prot, 'Skalierung fuer GNUPLOT: ', PlotScale);
writeln('Approximationsintervall = ', intval(Left, Right) );
writeln(Prot, 'Approximationsintervall = ', Left, '  ',  Right );
writeln('Funktionsauswertung ueber Gesamtintervall: ');
writeln(fEncl(intval(Left, Right)) );
writeln(Prot, 'Funktionsauswertung ueber Gesamtintervall: ');
writeln(Prot, fEncl(intval(Left, Right)) );
writeln(Prot, 'Entwicklungsstelle x0 = ', x0);
X:= Left;      { Linken Randpunkt erfassen }
ErrX:= Err(X); { Einschliessung des Fehlers am linken Randpunkt }
if RelErr then ErrX:= ErrX/fEncl(X);
InfErrX:= inf(ErrX);  { Initialisierung fuer repeat Schleife }
MidErrX:= mid(ErrX);  { Initialisierung fuer repeat Schleife }
SupErrX:= sup(ErrX);  { Initialisierung fuer repeat Schleife }
MiniX:=X;
Mini:=   inf(ErrX);
MiniUb:= sup(ErrX);
MaxiX:= X;
Maxi:=   sup(ErrX);
MaxiLb:= inf(ErrX);
writeln('Mini: ', Mini, '   Maxi: ', Maxi);
writeln(Prot, 'Mini: ', Mini, '   Maxi: ', Maxi);

X:= Right;     { Rechten Randpunkt erfassen }
ErrX:= Err(X); { Einschliessung des Fehlers am rechten Endpunkt }
if RelErr then ErrX:= ErrX/fEncl(X);
if Mini > inf(ErrX) then begin
  MiniX:= X;
  Mini:= inf(ErrX);
  MiniUb:= sup(ErrX);
end;
if Maxi < sup(ErrX) then begin
  MaxiX:= X;
  Maxi:= sup(ErrX);
  MaxiLb:= inf(ErrX);
end;
writeln(      'Mini: ', Mini, '   Maxi: ', Maxi);
writeln(Prot, 'Mini: ', Mini, '   Maxi: ', Maxi);
```

```
write('Anzahl von Teilintervallen : ');
 read(AnzTeilInt); writeln;         { AnzTeilInt:= 10000000; }
if AnzTeilInt < 2048 then AnzTeilInt:= 2048;
writeln('Gewaehlte Anzahl von Teilintervallen: ', AnzTeilInt);
writeln(Prot, 'Gewaehlte Anzahl von Teilintervallen: ', AnzTeilInt);

h:= (right-left)/AnzTeilInt;
                  { Breite eines Teilintervalls (Fenster) }
writeln('Breite eines Teilintervalls: ', h);
writeln(Prot, 'Breite eines Teilintervalls: ', h);

X:= Left; { Fenster ueber gesamtes Intervall bewegen }
Anz:= 0;
TeilIntProPixel:= AnzTeilInt div 512;
repeat
  if sup(X) > Right then X:= intval(inf(X), Right);
  ErrX:= Err(X); { Einschliessung des Fehlers im aktuellen  }
                 {    Teilintervall X.                     }
  if RelErr then ErrX:= ErrX/fEncl(X);
  { Achtung!  Skalierung der Werte der Fehlerkurve! }
  if Anz  mod TeilIntProPixel = 0 then begin
     writeln( out1, mid(x), '  ', PlotScale*InfErrX );
     writeln( out2, mid(x), '  ', PlotScale*MidErrX );
     writeln( out3, mid(x), '  ', PlotScale*SupErrX );

     write( outall, mid(x), '  ', PlotScale*InfErrX );
     write( outall, '  ', PlotScale*MidErrX );
     writeln( outall,'  ', PlotScale*SupErrX );
     InfErrX:= inf(ErrX);
     MidErrX:= mid(ErrX);
     SupErrX:= sup(ErrX);
  end else begin
     if InfErrX > inf(ErrX) then InfErrX:= inf(ErrX);
     if SupErrX < sup(ErrX) then SupErrX:= sup(ErrX);
  end;

  if Mini > inf(ErrX) then begin
    MiniX:= X;
    Mini:= inf(ErrX);
    MiniUb:= sup(ErrX);
  end;
  if Maxi < sup(ErrX) then begin
```

```
      MaxiX:= X;
      Maxi:= sup(ErrX);
      MaxiLb:= inf(ErrX);
    end;
    Anz:= Anz+1;
    if Anz mod 100000 = 0 then
      writeln(Anz:9, ' Mini= ', Mini, '  Maxi= ', Maxi);
    X:= intval(sup(X), sup(X)+h);
    if (inf(X) >= Right) then X:= Right;
  until X = Right;
            { Rechter Rand des Approx.-Intervalls ist erreicht }
  writeln('Endergebnis:  Mini: ', Mini,   '      Maxi: ', Maxi);
  writeln('            MiniUb: ', MiniUb, '    MaxiLb: ', MaxiLb);
  writeln(Prot, 'Endergebnis:  Mini: ', Mini,   '      Maxi: ', Maxi);
  writeln(Prot, '            MiniUb: ', MiniUb, '    MaxiLb: ', MaxiLb);
  writeln(Prot, ' MiniX: ', MiniX);
  writeln(Prot, ' MaxiX: ', MaxiX);
end.
{------------------------------------------------------------------}
```

Das in Abschnitt 3.3 angegebene Schaubild kann mit Hilfe von Gnuplot aus der vom obigen Programm erzeugten Datei `jzero_5847.gnu` mittel der Befehlsequenz

```
set output 'schaubild.ps'
set term postscript eps
set title 'JZERO 5847'
plot [0:64] 'jzero_5847.gnu' using 1:2 with lines,
            'jzero_5847.gnu' using 1:3 with lines,
            'jzero_5847.gnu' using 1:4 with lines
```

erstellt werden.

B.3 Gleichungssystemlöser für Intervall–Bandmatrizen

Die Implementierung des in Abschnitt 1.5 vorgestellten Gleichungssystemlösers für Systeme mit Bandmatrizen als Pascal–XSC Modul `bandlss` wird nachfolgend angegeben:

```
module bandlss;
```

```
use i_ari, mv_ari, mvi_ari;

procedure traceback (var t:text) ; external f_back ;
const   stderr = 2 ; { standard error }
procedure rewrite (var t : text ; nr : integer) ; external f_rwrn ;
procedure exit    (retcode : integer)           ; external a_exit ;

global  { BANDErrMsg must be global  }
procedure BANDErrMsg (errnr : integer) ;
  var message : text;
  begin
    rewrite(message,stderr);
    case  errnr  of
      0: writeln (message,'No errors') ;
      1: writeln (message,'read_Ab: Index range of band matrix
                          must be [1..n,-l..k]; l,k >= 0; l+k > 0;') ;
      2: writeln (message,'read_Ab: Bandwidth(s) too large') ;
      3: writeln (message,'read_Ab: Index range of right hand side
                          must be [1..n]') ;
      else: writeln (message,'Unknown error') ;
    end ;
    traceback (message) ;
    exit(-1);
  end ;

function min(a,b: integer): integer;
begin
  if a < b then min := a else min := b
end;

function max(a,b: integer): integer;
begin
  if a > b then max := a else max := b
end;

function max(a,b,c: integer): integer;
begin
  if a > b then max := max(a,c) else max := max(b,c);
end;

function max(a,b: real): real;
begin
  if a > b then max := a else max := b
```

```
end;

function sqr_b(x : real) : real;
begin
  sqr_b := x*x
end;

function norm(var A : rmatrix): real;
{ compute row-sum norm of A }
var i,j : integer;
   m,s : real;
   v : rvector[lb(A,2)..ub(A,2)];
begin
   m := 0.0;
   for i := lb(A) to ub(A) do
   begin
      For j := lb(A,2) to ub(A,2) do v[j] := abs(A[i,j]);
      s := #>( For j:=lb(A,2) to ub(A,2) SUM( v[j] ) );
      m := max(m,s);
   end;
   norm := m;
end; { norm }

function norm(var A : imatrix): real;
{ compute row-sum norm of A }
var i,j : integer;
   m,s : real;
   v : rvector[lb(A,2)..ub(A,2)];
begin
   m := 0.0;
   for i := lb(A) to ub(A) do
   begin
      For j := lb(A,2) to ub(A,2) do v[j] := sup(abs(A[i,j]));
      s := #>( For j:=lb(A,2) to ub(A,2) SUM( v[j] ) );
      m := max(m,s);
   end;
   norm := m;
end; { norm }

procedure inv_orth ( var a: rmatrix;      { Eingangsgroesse }
                     var a_inv: imatrix;  { Ausgangsgroesse }
                     var err: integer);   { Ausgangsgroesse }
{ Compute enclosure of the inverse of an almost orthogonal matrix a }
```

```
{ err = 0 : no errors;  err = 1 : a is not sufficient orthogonal    }
var nrm   : real;
    delta : interval;
    i,j   : integer;
    at    : rmatrix[lb(a,1)..ub(a,1),lb(a,2)..ub(a,2)];
begin
  err := 0;
  if ub(a,1) - lb(a,1) = 0 then a_inv := intval(1)/a[lb(a,1),lb(a,2)]
                           else
  if ub(a,1) - lb(a,1) = 1 then
  begin  { 2x2 matrix }
    delta := ##(   a[lb(a,1),lb(a,2)] * a[ub(a,1),ub(a,2)]
                 - a[lb(a,1),ub(a,2)] * a[ub(a,1),lb(a,2)] );
    if 0 in delta then err := 1 else
    begin
      a_inv[lb(a_inv,1),lb(a_inv,2)] :=  a[ub(a,1),ub(a,2)] / delta;
      a_inv[lb(a_inv,1),ub(a_inv,2)] := -a[lb(a,1),ub(a,2)] / delta;
      a_inv[ub(a_inv,1),lb(a_inv,2)] := -a[ub(a,1),lb(a,2)] / delta;
      a_inv[ub(a_inv,1),ub(a_inv,2)] :=  a[lb(a,1),lb(a,2)] / delta;
    end;
  end                      else
  begin
    at := transp(a);
    nrm := norm( ##(id(a) - at*a) );
    if nrm < 1 then
    begin
      nrm := norm(at) *> nrm /> (1 -< nrm);
      delta := intval(-nrm,nrm);
      for i := lb(a,1) to ub(a,1) do
      for j := lb(a,2) to ub(a,2) do a_inv[i,j] := at[i,j] + delta;
    end        else
    begin
      err := 1;
      a_inv := at
    end;
  end;
end;

function QR(var A: rmatrix): rmatrix[1..ub(A),1..ub(A)];
{ Perform QR-decomposition of A by Householder matrices }
var i,j,k,n : integer;
    Q       : rmatrix[1..ub(A),1..ub(A)];
    r,s     : real;
```

```
    b,c,d   : rvector[1..ub(A)];
begin
  n := ub(A);
  Q := id(Q);
  For k := 1 to n-1 do
  begin
    b[k..n] := A[k..n,k];
    if rvector(b[k..n]) <> NULL(b[k..n]) then
    begin
      s := sqrt(b*b);
      if sign(A[k,k]) >= 0 then s := -s;
      b[k] := A[k,k] - s;
      s := 1 / (s * b[k]);
      For i := k+1 to n do
      begin
        r := s * #*( For j:= k to n SUM( b[j]*A[j,i] ) );
        For j := k to n do A[j,i] := A[j,i] + b[j] * r;
      end;
      For i := 1 to n do
      begin
        r := s * #*( For j:= k to n SUM( b[j]*Q[i,j] ) );
        For j := k to n do Q[i,j] := Q[i,j] + b[j] * r;
      end;
    end;
    b[k] := 0;
  end;
  QR := Q;
end;

function QR(var A: rmatrix; var y: ivector):
                                  rmatrix[1..ub(A),1..ub(A)];
var i,j,k,n   : integer;
    hv,laenge : rvector[1..ub(A)];
    hr        : real;
begin
  n := ub(A);
  For i := 1 to n do
    laenge[i] := sqr_b(diam(y[i]))*(rvector(A[*,i])*rvector(A[*,i]));
  For j := 1 to n-1 do
  For k := j+1 to n do
  begin
    if laenge[j] < laenge[k] then
    begin
```

```
      hr := laenge[k];  laenge[k] := laenge[j];   laenge[j] := hr;
      hv := rvector(A[*,k]);  A[*,k] := A[*,j];   A[*,j] := hv;
    end;
  end;
  QR := QR(A);
end;

procedure lss_triangular(var A: rmatrix; var b,x: rvector;
                         var Err: integer);
{ Floating point forward substitution for lower triangular system   }
{ Ax=b or Floating point backward substitution for upper triangular }
{ system Ax=b.                                                      }
{ A must be in [1..n,-l..0]  or  [1..n,0..k]                        }
{ Err = 0:  No errors;  Err = 2: A is not a triangular matrix       }
var i,j,k,n,l : integer;
begin
  n := ub(A,1);  Err := 0;
  l := abs( LB(A,2) );
  k := ub(A,2);
  if k = 0 then { A is lower triangular matrix }
  For i := 1 to n do
    x[i] := #*( b[i] - For j:= max(1,i-l) to i-1
                SUM(A[i,j-i]*x[j]) ) / A[i,0]
          else
  if l = 0 then { A is upper triangular matrix }
  For i := n downto 1 do
    x[i] := #*( b[i] - For j:= i+1 to min(n,i+k)
                SUM(A[i,j-i]*x[j]) ) / A[i,0]
          else err := 2; { A is not a triangular matrix ! }
end;

type coefficient_matrix  = rvector;
type coefficient_imatrix = ivector;

operator * (var a: coefficient_matrix; var b: rmatrix)
            res: imatrix[lb(b,1)..ub(b,1),lb(b,2)..ub(b,2)];
var i,j,n : integer;
begin
  n := ub(b);
  For i := 1 to n-1 do res[i] := b[i+1];
  For j := 1 to n do
             res[n,j] := ##( For i := 1 to n SUM(a[i]*b[i,j]) );
end;
```

```
operator * (var a: coefficient_imatrix; var b: rmatrix)
            res: imatrix[lb(b,1)..ub(b,1),lb(b,2)..ub(b,2)];
var i,j,n : integer;
begin
  n := ub(b);
  For i := 1 to n-1 do res[i] := b[i+1];
  For j := 1 to n do
             res[n,j] := ##( For i := 1 to n SUM(a[i]*b[i,j]) );
end;

procedure lss_lower(var A: rmatrix; var b,x: ivector; l: integer;
                    var err: integer);
{ Forward substitution using coordinate transformations }
{ err = 0: No errors;  err = 1: Matrix inversion failed }
var i,j,n    : integer;
    af       : coefficient_matrix[1..l];
    c0,c1    : rmatrix[1..l,1..l];
    c_inv,ai : imatrix[1..l,1..l];
    y0,y1,bi : ivector[1..l];
begin
  n := ub(A,1);  err := 0;
  c0 := id(c0);
  bi := 0;
  For i := 1 to l do
  begin
    x[i] := ##( b[i] - For j:= 1 to i-1 SUM(A[i,j-i]*x[j]) )/A[i,0];
    y0[i] := x[i];
  end;
  For i := l+1 to n do
  begin
    For j := 1 to l do af[j] := -A[i,j-l-1] / A[i,0];
    bi[l] := b[i] / A[i,0];
    ai := af * c0;
    c1 := QR( mid(ai),y0);
    inv_orth(c1,c_inv,err);
    y0 := (c_inv * ai) * y0 + c_inv * bi;
    x[i] := c1[l] * y0;
    c0 := c1;
  end
end;

procedure lss_lower(var A: imatrix; var b,x: ivector; l: integer;
```

```
                    var err: integer);
{ Forward substitution using coordinate transformations }
{ err = 0: No errors;  err = 1: Matrix inversion failed }
var i,j,n    : integer;
    af       : coefficient_imatrix[1..l];
    c0,c1    : rmatrix[1..l,1..l];
    c_inv,ai : imatrix[1..l,1..l];
    y0,y1,bi : ivector[1..l];
begin
  n := ub(A,1);  err := 0;
  c0 := id(c0);
  bi := 0;
  For i := 1 to l do
  begin
    x[i] := ##( b[i] - For j:= 1 to i-1 SUM(A[i,j-i]*x[j]) )/A[i,0];
    y0[i] := x[i];
  end;
  For i := l+1 to n do
  begin
    For j := 1 to l do af[j] := -A[i,j-l-1] / A[i,0];
    bi[l] := b[i] / A[i,0];
    ai := af * c0;
    c1 := QR( mid(ai),y0);
    inv_orth(c1,c_inv,err);
    y0 := (c_inv * ai) * y0 + c_inv * bi;
    x[i] := c1[l] * y0;
    c0 := c1;
  end
end;

procedure lss_upper(var A: rmatrix; var b,x: ivector; k: integer;
                    var err: integer);
{ Backward substitution using coordinate transformations }
{ err = 0: No errors;  err = 1: Matrix inversion failed }
var i,j,n    : integer;
    af       : coefficient_matrix[1..k];
    c0,c1    : rmatrix[1..k,1..k];
    c_inv,ai : imatrix[1..k,1..k];
    y0,y1,bi : ivector[1..k];
begin
  n := ub(A,1);  err := 0;
  c0 := id(c0);
  bi := 0;
```

```
  for i := n downto n-k+1 do
  begin
    x[i] := ##( b[i] - For j:= i+1 to n SUM(A[i,j-i]*x[j]) ) / A[i,0];
    y0[n+1-i] := x[i];
  end;
  For i := n-k downto 1 do
  begin
    For j := 1 to k do af[j] := -A[i,k+1-j] / A[i,0];
    bi[k] := b[i] / A[i,0];
    ai := af * c0;
    c1 := QR( mid(ai),y0);
    inv_orth(c1,c_inv,err);
    y0 := (c_inv * ai) * y0 + c_inv * bi;
    x[i] := c1[k] * y0;
    c0 := c1;
  end;
end;

procedure lss_upper(var A: imatrix; var b,x: ivector; k: integer;
                    var err: integer);
{ Backward substitution using coordinate transformations }
{ err = 0: No errors;  err = 1: Matrix inversion failed }
var i,j,n    : integer;
    af       : coefficient_imatrix[1..k];
    c0,c1    : rmatrix[1..k,1..k];
    c_inv,ai : imatrix[1..k,1..k];
    y0,y1,bi : ivector[1..k];
begin
  n := ub(A,1);  err := 0;
  c0 := id(c0);
  bi := 0;
  for i := n downto n-k+1 do
  begin
    x[i] := ##( b[i] - For j:= i+1 to n SUM(A[i,j-i]*x[j]) ) / A[i,0];
    y0[n+1-i] := x[i];
  end;
  For i := n-k downto 1 do
  begin
    For j := 1 to k do af[j] := -A[i,k+1-j] / A[i,0];
    bi[k] := b[i] / A[i,0];
    ai := af * c0;
    c1 := QR( mid(ai),y0);
    inv_orth(c1,c_inv,err);
```

```
    y0 := (c_inv * ai) * y0 + c_inv * bi;
    x[i] := c1[k] * y0;
    c0 := c1;
  end;
end;

procedure lss_triangular(var A: rmatrix; var b,x: ivector;
                         var err: integer);
{ Interval forward substitution for lower triangular system Ax=b or }
{ Interval backward substitution for upper triangular system Ax=b.  }
{ A must be in [1..n,-l..0]  or  [1..n,0..k]                        }
{ err = 0: No errors;  err = 1: Matrix inversion failed             }
{ err = 2: A is not a regular triangular matrix !                   }
var i,j,k,n,l : integer;
begin
   n := ub(A,1);  err := 0;
   l := abs( LB(A,2) );
   k := ub(A,2);
   if k = 0 then lss_lower(A,b,x,l,err){ A: lower triangular matrix }
            else
   if l = 0 then lss_upper(A,b,x,k,err){ A: upper triangular matrix }
   else err := 2;          { A is not a regular triangular matrix ! }
end;

procedure lss_triangular(var A: imatrix; var b,x: ivector;
                         var err: integer);
{ Interval forward substitution for lower triangular system Ax=b or }
{ Interval backward substitution for upper triangular system Ax=b.  }
{ A must be in [1..n,-l..0]  or  [1..n,0..k]                        }
{ err = 0: No errors;  err = 1: Matrix inversion failed             }
{ err = 2: A is not a regular triangular matrix !                   }
var i,j,k,n,l : integer;
begin
  n := ub(A,1);  err := 0;
  l := abs( LB(A,2) );
  k := ub(A,2);
  if k = 0 then lss_lower(A,b,x,l,err) { A: lower triangular matrix }
           else
  if l = 0 then lss_upper(A,b,x,k,err) { A: upper triangular matrix }
  else err := 2;           { A is not a regular triangular matrix ! }
end;

procedure lu_decomp(var A,Lo,Up: rmatrix; var LU_A: imatrix);
```

```
{ Compute approximate LU-decomposition of A without pivoting,       }
{ store factors in Lo,Up and store enclosure of Defekt LU-A in LU_A }
{ A and LU_A must be  [1..n,-l..k]  }
{ Lo         must be  [1..n,-l..0]  }
{ Up         must be  [1..n, 0..k]  }
var i,j,m,n,k,l : integer;
    dot         : dotprecision;
begin
  n := ub(A,1);
  l := abs( lb(A,2) );
  k := ub(A,2);
  For i := 1 to n do
  begin
    Lo[i,0] := 1;
    For j := i to min(n,i+max(k,l)) do
    begin
      if j-i <= k then
      begin
        dot := #( A[i,j-i] - for m:= max(1,i-l,j-k) to min(i-1,j)
                  SUM( Lo[i,m-i]*Up[m,j-m] ) );
        Up[i,j-i] := #*( dot );
        LU_A[i,j-i] := ##(Up[i,j-i] - dot );
      end;
      if ( sign(Up[i,0]) <> 0 ) and ( i <> j ) and ( j-i <= l ) then
      begin
        dot := #( A[j,i-j] - for m:= max(1,j-l,i-k) to min(j,i-1)
                  SUM( Lo[j,m-j]*Up[m,i-m] ) );
        Lo[j,i-j] := #*( dot ) / Up[i,0];
        LU_A[j,i-j] := ##(Lo[j,i-j]*Up[i,0] - dot );
      end;
    end;
 end;
end;

procedure LU_decomp(var A: imatrix; var Lo,Up: rmatrix;
                    var LU_A: imatrix);
{ Compute approximate LU-decomposition of A without pivoting,       }
{ store factors in Lo,Up and store enclosure of Defekt LU-A in LU_A }
{ A and LU_A must be  [1..n,-l..k]  }
{ Lo         must be  [1..n,-l..0]  }
{ Up         must be  [1..n, 0..k]  }
var i,j,m,n,k,l : integer;
    dot : dotprecision;
```

```
    x : real;
begin
  n := ub(A,1);
  l := abs( lb(A,2) );
  k := ub(A,2);
  For i := 1 to n do
  begin
    Lo[i,0] := 1;
    For j := i to min(n,i+max(k,l)) do
    begin
      if j-i <= k then
      begin
         dot := #( for m:= max(1,i-l,j-k) to min(i-1,j)
                   SUM( Lo[i,m-i]*Up[m,j-m] ) );
 x := mid(A[i,j-i]);
 Up[i,j-i] := #*(x - dot);
 LU_A[i,j-i] := ##(Up[i,j-i] - A[i,j-i] + dot );
      end;
      if ( sign(Up[i,0]) <> 0 ) and ( i <> j ) and ( j-i <= l ) then
      begin
         dot := #( for m:= max(1,j-l,i-k) to min(j,i-1)
                    SUM( Lo[j,m-j]*Up[m,i-m] ) );
 x := mid(A[j,i-j]);
 Lo[j,i-j] := #*(x - dot) / Up[i,0];
         LU_A[j,i-j] := ##(Lo[j,i-j]*Up[i,0] - A[j,i-j] + dot );
      end;
    end;
 end;
end;

global procedure lss(var A: rmatrix; b: rvector;
                     var x: ivector;     { Ausgangsgroesse }
                     var err: integer);  { Ausgangsgroesse }
{ Solving   Ax=b;  A with band structure                  }
{ Index range of A must be:     [1..n,-l..k]  with:       }
{ l,k = 0,1,2,..  and   l+k > 0  and  l,k < n;            }

{ err = 0:   No errors;                                   }
{ err = 1:   Intern matrix inversion failed;              }
{ err = 2:   An intern matrix is not triangular           }
{ err = 3:   Inclusion failed  --->  solution not unique  }
{ err = 4:   A has no correct shape                       }
{ err = 5:   Bandwidths too large                         }
```

```
Label 1;
const eps = 0.1;
var Lo          : rmatrix[lb(A,1)..ub(A,1),lb(A,2)..0      ];
    Up          : rmatrix[lb(A,1)..ub(A,1),      0..ub(A,2)];
    LU_A        : imatrix[lb(A,1)..ub(A,1),lb(A,2)..ub(A,2)];
    b_app,x_app : rvector[lb(A,1)..ub(A,1)];
    defect,z,za : ivector[lb(A,1)..ub(A,1)];
    i,j,k,l,m,n : integer;
    enthalten   : boolean;
begin
  n := ub(A,1);  err := 0;
  l := abs( lb(A,2) );
  k := ub(A,2);
  if (k<0) or (lb(A,2)>0) or ( (l=0) and (k=0) ) then
  begin  err := 4; goto 1 end;
  if (l>=n) or (k>=n) then
  begin  err := 5; goto 1 end;
  if (l=0) or (k=0) then
  begin
    lss_triangular(A,b,x_app,err);
    if err <> 0 then goto 1;
    For i := 1 to n do
      defect[i] := ##( b[i] - For j:= max(1,i-l) to min(n,i+k)
                      SUM( A[i,j-i]*x_app[j] )  );
    lss_triangular(A,defect,z,err);
    if err <> 0 then goto 1;
    x := x_app + z;
  end               else
  begin
    { Compute LU-factorization and Defect LU-A : }
    lu_decomp(A,Lo,Up,LU_A);
    { Compute approximate solution x_app: }
    lss_triangular(Lo,b,b_app,err);
    if err <> 0 then goto 1;
    lss_triangular(Up,b_app,x_app,err);
    if err <> 0 then goto 1;
    { Compute defect := b - A*x_app  of approx. solution x_app : }
    For i := 1 to n do
      defect[i] := ##( b[i] - For j:= max(1,i-l) to min(n,i+k)
                      SUM( A[i,j-i]*x_app[j] )  );
    z := defect;
    m := 0;  { Iteration until inclusion is obtained }
```

```
    repeat   { or max. iteration count exceeded      }
      za := blow(z,eps);
      for i := 1 to n do
        z[i] := ##( defect[i] + For j:= max(1,i-l) to min(n,i+k)
                    SUM( LU_A[i,j-i]*za[j] )  );
      lss_triangular(Lo,z,x,err);
      if err <> 0 then goto 1;
      lss_triangular(Up,x,z,err);
      if err <> 0 then goto 1;
      enthalten := z in za;
      m := m + 1;
    until enthalten or (m = 10);
    if enthalten then x := x_app + z
                 else err := 3;
  end;
1:
end;

global procedure lss(var A: imatrix; b: ivector;  { Eingangsgroessen }
                     var x: ivector;     { Ausgangsgroesse }
                     var err: integer);  { Ausgangsgroesse }
{ Solving   Ax=b;  A with band structure                }
{ Index range of A must be:     [1..n,-l..k]  with:     }
{ l,k = 0,1,2,..  and   l+k > 0  and  l,k < n;          }

{ err = 0:   No errors;                                 }
{ err = 1:   Intern matrix inversion failed;            }
{ err = 2:   An intern matrix is not triangular         }
{ err = 3:   Inclusion failed  --->  solution not unique }
{ err = 4:   A has no correct shape                     }
{ err = 5:   Bandwidths too large                       }

Label 1;
const eps = 0.1;
var Lo          : rmatrix[lb(A,1)..ub(A,1),lb(A,2)..0      ];
    Up          : rmatrix[lb(A,1)..ub(A,1),      0..ub(A,2)];
    LU_A        : imatrix[lb(A,1)..ub(A,1),lb(A,2)..ub(A,2)];
    b_app,x_app : rvector[lb(A,1)..ub(A,1)];
    defect,z,za : ivector[lb(A,1)..ub(A,1)];
    i,j,k,l,m,n : integer;
    enthalten   : boolean;
begin
  n := ub(A,1);  err := 0;
```

```
l := abs( lb(A,2) );
k := ub(A,2);
if (k<0) or (lb(A,2)>0) or ( (l=0) and (k=0) ) then
begin  err := 4; goto 1 end;
if (l>=n) or (k>=n) then
begin  err := 5; goto 1 end;
if (l=0) or (k=0) then
begin
   lss_triangular(mid(A),mid(b),x_app,err);
   if err <> 0 then goto 1;
   For i := 1 to n do
     defect[i] := ##( b[i] - For j:= max(1,i-l) to min(n,i+k)
                      SUM( A[i,j-i]*x_app[j] )  );
   lss_triangular(A,defect,z,err);
   if err <> 0 then goto 1;
   x := x_app + z;
end               else
begin
   { Compute LU-factorization and Defect LU-A : }
   lu_decomp(A,Lo,Up,LU_A);
   { Compute approximate solution x_app: }
   lss_triangular(Lo,mid(b),b_app,err);
   if err <> 0 then goto 1;
   lss_triangular(Up,b_app,x_app,err);
   if err <> 0 then goto 1;
   { Compute defect := b - A*x_app  of approx. solution x_app : }
   For i := 1 to n do
     defect[i] := ##( b[i] - For j:= max(1,i-l) to min(n,i+k)
                      SUM( A[i,j-i]*x_app[j] )  );
   z := defect;
   m := 0;  { Iteration until inclusion is obtained }
   repeat   { or max. iteration count exceeded      }
     za := blow(z,eps);
     for i := 1 to n do
       z[i] := ##( defect[i] + For j:= max(1,i-l) to min(n,i+k)
                    SUM( LU_A[i,j-i]*za[j] )  );
     lss_triangular(Lo,z,x,err);
     if err <> 0 then goto 1;
     lss_triangular(Up,x,z,err);
     if err <> 0 then goto 1;
     enthalten := z in za;
     m := m + 1;
   until enthalten or (m = 10);
```

```
     if enthalten then x := x_app + z
                  else err := 3;
  end;
1:
end;

global procedure read_Ab(var A: rmatrix; var b: rvector);
{ read matrix A and right hand side vector b             }
{ Index range of A must be:     [1..n,-l..k]  with:      }
{ l,k = 0,1,2,..  and   l+k > 0  and  l,k < n;           }
{ Index range of b must be:     [1..n]                   }
var i,j,k,l,m : integer;
    c         : char;
begin
  { read A: }
  A := NULL(A);  { Initialisation }
  l := abs( lb(A,2) );
  k := ub(A,2);
  if (k<0) or (lb(A,2)>0) or ( (k=0) and (l=0) ) or (LB(A)<>1)
                                          then BANDErrMsg(1);
  if (l>=ub(A)) or (k>=ub(A)) then BANDErrMsg(2);
  if (lb(b)<>1) or (ub(b)<>ub(A)) then BANDErrMsg(3);

   writeln(l+k+1:2,' band values
                  (start with const. value of lowest band) of A = ?');
  For j := -l to k do
  begin
    read( a[max(1,1-j),j] );
    For i := 2 to ub(A,1)-abs(j) do a[max(i,i-j),j] := a[max(1,1-j),j];
  end;
  { Allow for changes of individual elements of A: }
  write('change individual band elements of A ? (y/n) ');
  readln;  read(c);
  if c in ['j','J','y','Y'] then
  repeat
    writeln('row = 0   or  col = 0   finish the change
                                          of individual elements');
    write('row, col, new value : ');  read(i,j);
    if (i>0) and (j>0) then read(a[i,j-i]);
  until (i=0) or (j=0);

  { read b: }
  writeln(' b = ?');
```

```
  writeln(' input of  = reAL-value     <enter>  ===>  b[i] = const.');
  writeln(' input of  value1 value2 .. <enter>  ===>  different b[i]');
  while eoln or (input^=' ') do get(input);
  if input^ = '=' then
  begin
    { set all elements of b equal to b[1]: }
    get(input);
    read( b[1] );
    For i := 2 to ub(A,1) do b[i] := b[1];
  end             else  { read each b[i] separately: }
  For i := 1 to ub(A,1) do read( b[i] );

  { Allow for changes of individual elements of b: }
  write('change individual elements of b ? (y/n) ');
  readln;  read(c);
  if c in ['j','J','y','Y'] then
  repeat
    writeln('row = 0   finish the change of individual elements');
    write('row, new value : ');  read(i);
    if i > 0 then read(b[i]);
  until i = 0 ;
end;

global procedure read_Ab(var A: imatrix; var b: ivector);
{ read matrix A and right hand side vector b             }
{ Index range of A must be:     [1..n,-1..k]  with:      }
{ l,k = 0,1,2,..  and   l+k > 0  and  l,k < n;           }
{ Index range of b must be:     [1..n]                   }
var i,j,k,l,m : integer;
    c         : char;
begin
  { read A: }
  A := NULL(A);  { Initialisation }
  l := abs( lb(A,2) );
  k := ub(A,2);
  if (k<0) or (lb(A,2)>0) or ( (k=0) and (l=0) ) or (LB(A)<>1)
                                          then BANDErrMsg(1);
  if (l>=ub(A)) or (k>=ub(A)) then BANDErrMsg(2);
  if (lb(b)<>1) or (ub(b)<>ub(A)) then BANDErrMsg(3);

  writeln(l+k+1:2,' interval band values
                 (start with const. value of lowest band) of A = ?');
  For j := -l to k do
```

```
  begin
    read( a[max(1,1-j),j] );
    For i := 2 to ub(A,1)-abs(j) do a[max(i,i-j),j] := a[max(1,1-j),j];
  end;
  { Allow for changes of individual interval elements of A: }
  write('change individual band elements of A ? (y/n) ');
  readln;  read(c);
  if c in ['j','J','y','Y'] then
  repeat
    writeln('row = 0   or  col = 0   finish the change
                                              of individual elements');
    write('row, col, new interval value : ');  read(i,j);
    if (i>0) and (j>0) then read(a[i,j-i]);
  until (i=0) or (j=0);

  { read b: }
  writeln(' b = ?');
  writeln(' input of  = interval      <enter>  ===>  b[i] = const.');
  writeln(' input of  intv1 intv2 ... <enter>  ===>  different b[i]');
  while eoln or (input^=' ') do get(input);
  if input^ = '=' then
  begin
    { set all elements of b equal to b[1]: }
    get(input);
    read( b[1] );
    For i := 2 to ub(A,1) do b[i] := b[1];
  end             else  { read each b[i] separately: }
  For i := 1 to ub(A,1) do read( b[i] );

  { Allow for changes of individual interval elements of b: }
  write('change individual elements of b ? (y/n) ');
  readln;  read(c);
  if c in ['j','J','y','Y'] then
  repeat
    writeln('row = 0   finishs the change of individual elements');
    write('row, new interval value : ');  read(i);
    if i > 0 then read(b[i]);
  until i = 0 ;
end;

begin

end.
```

B.4 Referenzfunktion

Eine Implementierung der in Abschnitt 5.2 vorgestellten Referenzfunktion kann folendermassen aussehen:

```
function j0( x:mpreal ): mpinterval;
var arg,argsqr,arghalb,r,p:mpinterval;
    i,n,vz:integer;
    pinterval:interval;
begin
  mpinit(arg); mpinit(argsqr); mpinit(arghalb); mpinit(r); mpinit(p);

  arg:=x;
  argsqr:=sqr(x);
  arghalb:=argsqr/4.0;

  r:=1;
  n:=0;
  while (r.sup>1e-17) do begin
    n:=n+10;
    r:= power(arghalb,_mpinterval(n))
         / sqr(fakultaet(n));
    if (r.sup-r.inf)>1e-18 then begin
       setprec(getprec+2);
       n:=n-10;
       writeln('*** Precision = ',getprec);
    end;
writeln(n,r);
  end;
  r.inf:=-r.sup;
  writeln;
  writeln('Approximationsgrad: ',n);
  pinterval:=r;
  writeln('Restgliedeinschliessung: ',pinterval);
  writeln;

  if odd(n) then vz:=-1 else vz:=1;

  p:=vz/(sqr(fakultaet(n)));
  for i:=n-1 downto 0 do begin
    vz:=-vz;
    p:=p*arghalb + vz/(sqr(fakultaet(i)));
```

```
    pinterval:=p;
    writeln(i:4,': ',pinterval);
  end;
  writeln;

  j0:=true;
  j0:=p+r;
  j0:=false;
  mpfree(arg); mpfree(argsqr); mpfree(arghalb); mpfree(r); mpfree(p);
end;
```

B.5 Miller–Algorithmus

Eine Pascal–XSC Implementierung des in Abschnitt 5.4.6.2 erläuterten Miller–Algorithmus ist hier angegeben:

```
program miller;

var x,s : real;
    i,n : integer;
    p   : dynamic array [*] of real;  { Dynamisches Feld }

begin
  write('x = '); read(x);
  write('n = '); read(n);

  { Miller- Algorithmus, hier nur Schritte 1 - 5 implementiert }
  allocate( p, 0..n+1 );              { Speicherallokierung }
  p[n+1]:= 0;                         { Setzen der Startwerte }
  p[ n ]:= 1;

  if odd(n) then s:= 0 else s:= 1;
  for i:= n-1 downto 0 do begin
    p[i]:= 2*(i+1)/x * p[i+1]  -  p[i+2];
    if (i>0) and not odd(i) then
                        s:= s + p[i]; { Summe fuer Normierung }
  end;
  s:= 2*s + p[0];

  { Normierung und Ergebnisausgabe }
  for i:= n+1 downto 0 do  writeln( i:4, '  ', p[i] / s );
end.
```

Aus Gründen der Einfachheit und Übersichtlichkeit wurde in dieser Implementierung die Iteration (Schritt 6) aus Algorithmus Miller nicht implementiert.

B.6 Verwendung einer erzeugenden Funktion

Ein Programm zur Berechnung von Einschließungen für Besselfunktionen mittels einer erzeugenden Funktion (vgl. Abschnitt 5.4.4) ist nachfolgend angegeben. Im Programm wird bei der Berechnung die von Pascal–XSC zur Verfügung gestellte komplexe Intervallarithmetik eingesetzt.

```
program ibessel_exp;

use i_ari, c_ari, ci_ari, mv_ari, mvi_ari, mvc_ari, mvci_ari;

function exp(z:cinterval):cinterval;
var x: interval;
begin
  x:= exp( re(z) );
  exp.re:= x * cos( im(z) );
  exp.im:= x * sin( im(z) );
end;

function power_factorial(x:interval; n:integer):interval;
var i : integer;
    f : interval;
begin
  f:= 1;
  for i:= 1 to n do f:= f*x/i;
  power_factorial:= f;
end;

var UnitRoots,BesselVector : civector;
    Bessel                 : ivector;
    i                      : complex;
    n,j,k                  : integer;
    z,Error,Pi2            : interval;


function BesselSolve(Bv:civector):ivector[-n..n];
var j,k,l  : integer;
    Bv_j   : cinterval;
```

```
begin
  for k:= -n to n do
  begin
    Bv_j:= 0;
    for j:= -n to n do
       Bv_j:= Bv_j + conj( UnitRoots[(j*k+n) mod (2*n+1) - n] ) * Bv[j];
    BesselSolve[k]:= re( Bv_j ) / (2*n+1) ;
  end;
end;

begin
  I  := compl(0,1);
  Pi2:= 8*arctan(intval(1));  { 2*pi }

  write('n = '); read(n);
  write('z = '); read(z);

  allocate( UnitRoots   , -n..n );
  allocate( BesselVector, -n..n );
  allocate( Bessel      , -n..n );

  for j:= -n to n do
     UnitRoots[j]:= exp( j*Pi2*I/(2*n+1) );

  Error:= 2*power_factorial( abs(z/2), n+1 )
                             * exp(abs(z/2)) * intval(-1,1);

  for j:= -n to n do
     BesselVector[j]:= exp( z*I*sin(j*Pi2/(2*n+1)) ) + Error;

  Bessel:= BesselSolve( BesselVector );

  writeln( 'Error = ', Error );
  writeln;

  for j:= 0 to n do
  begin
    writeln( j:3,'  ', Bessel[j]);
  end;
  writeln;
end.
```

B.7 Polynomdarstellung von Besselfunktionen höherer Ordnung

Auszüge aus dem Quelltext der benötigten Polynomarithmetik (vgl. Abschnitt 5.4.7) werden nachfolgend angegeben:

```
type polynom = rvector;

procedure init(var p:rvector);
var i:integer;
begin
  for i:=lb(p) to ub(p) do p[i]:=0;
end;

function x_mal(p:rvector):rvector;
var i:integer;
    erg:rvector;
begin
  allocate( erg, lb(p)..ub(p) );
  for i:=ub(p) downto (lb(p)+1) do erg[i]:=p[i-1];
  erg[lb(p)]:= 0;
  x_mal:=erg;

end;

operator * (a:integer; p:rvector) erg:rvector;
var i:integer;
begin
  allocate( erg, lb(p)..ub(p) );
  for i:=lb(p) to ub(p) do erg[i]:= a*p[i];
end;

operator - (p:rvector; q:rvector) erg:rvector;
var i:integer;
begin
  allocate( erg, lb(p)..ub(p) );
  for i:=lb(p) to ub(p) do erg[i]:= p[i]-q[i];
end;
```

Der folgende Programmausschnitt zeigt die Berechnung der Polynomwerte:

```
begin
```

```
write('(n>=1): n = '); read(n);

allocate( p_nm1, 0..n );
allocate( p_n  , 0..n );
allocate( p_np1, 0..n );
allocate( q_nm2, 0..n );
allocate( q_nm1, 0..n );
allocate( q_n  , 0..n );

init( p_nm1 );
init( p_n   );
init( p_np1 );
init( q_nm2 );
init( q_nm1 );
init( q_n   );

{ Def. p1 }
p_nm1[1]:= 1;

{ Def. p2 }
if (n>=2) then begin
   p_n[2]  := 2;
   p_n[0]  := -1;
end;

{ Def. q0 }
q_nm2[0]:= 1;

{ Def. q1 }
q_nm1[1]:= 2;

if (n=1) then begin
                 p_np1:= p_nm1;
                 q_n  := q_nm2;
              end;
if (n=2) then begin
                 p_np1:= p_n;
                 q_n  := q_nm1;
              end;
if (n>=3) then
   begin

      for i:= 3 to n do begin
```

```
            p_np1:=   i * x_mal(p_n)   - p_nm1;
            q_n  :=   i * x_mal(q_nm1) - q_nm2;

            p_nm1:= p_n;
            p_n  := p_np1;

            q_nm2:= q_nm1;
            q_nm1:= q_n;
         end;
      end;
```

B.8 Besselfunktion J_0

Nachfolgenden finden sich Auszüge aus der Implementierung der Funktion J_0 (vgl. Abschnitt 5.6) in der Programmiersprache ANSI–C. Bei Vergabe der Datei-, Funktions- und Variablennamen wurde die Namenskonvension der Bibliothek `FI_LIB` (vgl. Hofschuster/Krämer [51], [52]) und die des Laufzeitsystems (vgl. Cordes [32]) der XSC-Sprachen eingehalten. Hieraus ergeben sich die etwas kryptischen Bezeichnungen in den folgenden Quelltexten.

Die benötigten Prototypen finden sich im Headerfile `q_fcth.h`:

```
...
_PROTOTYPE(double q_j0,(double x));
_PROTOTYPE(a_intv j_j0,(double x));
...
```

Die Variablendefinitionen der globalen Konstanten und Koeffizienten befinden sich im Headerfile `q_defs.h`:

```
...
/* Koeffizienten fuer Zaehlerpolynom der Approximation in [0,6] */
extern double q_j0p6[5];
/* Koeffizienten fuer Nennerpolynom der Approximation in [0,6] */
extern double q_j0q6[3];
...
/* Gesamtfehlerschranke, Rundung nach oben  */
extern double q_j0p;
/* Gesamtfehlerschranke, Rundung nach unten */
extern double q_j0m;
...
```

In der Datei `q_glbl.c` werden unter anderem die Werte der Koeffizienten der Approximationen bereitgestellt:

```
...
   /* Koeffizienten der rationalen Approximation fuer J0(x), 0<=x<=6 */
   double q_j0p6[5]={   4.804031007812411E-003,
                       -1.690583302647642E-004,
                        2.288106856368657E-006,
                       -1.540585490435812E-008,
                        5.346523139404624E-011,
                       -7.945554339748063E-014};
   double q_j0q6[3]={   8.465710360960413E-001,
                        7.683641123494186E-003,
                        2.959776315003865E-005,
                        5.067975241046154E-008};
...
   /* Nullstellen der Funktion J0(x) */
   /* q_n1a = 2.404825557695773E+000;  */
   q_n1a = 5415185742764864.0 / 2251799813685248.0;
   /* q_n1b =-1.176691651530894E-016;  */
   q_n1b =-4773228410709390.0 / 40564819207303340847894502572032.0;
   /* q_n2a = 5.520078110286311E+000;  */
   q_n2a = 6215055430135365.0 / 1125899906842624.0;
   /* q_n2b = 8.088597146146723E-017;  */
   q_n2b = 6562249617483031.0 / 81129638414606681695789005144064.0;
...
```

Der eigentliche Kern der Routine befindet sich in der Datei `q_j0.c`:

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

#ifdef LINT_ARGS
local double q_j0(double x)
#else
local double q_j0(x)

double x;
#endif

{
  ...
  double res;
```

```
  double xx;
  ...

#ifdef PXSCTRAP
  E_SPUSH("q_j0");
#endif

  /* Special cases  */
  if NANTEST(x)                                   /* Test: x=NaN */
    res=q_abortnan(INV_ARG,&x,10);
  else {
    if (x==0) res = 1.0;         /* Sonderfall: x=0  =>  J_0(0)=1 */
    else {
      if (x<0) x = -x;        /* Beruecksichtigung neg. Argumente */

      /* Ab hier gilt: x>0 */
      if (x<=6)
        xx = x*x;
        /* rationale Approximation, Horner-Schema,      */
        /* aus Laufzeitgruenden ausprogrammiert,        */
        /* nicht als For-Schleife!                      */
        res = ((q_j0q6[3]*xx+q_joq6[2])*xx+q_j0q6[1])
                 *xx+q_j0q6[0];
        res = (((((q_j0p6[5]*xx+q_j0p6[4])*xx+q_j0p6[3])
                 *xx+q_jop6[2])*xx+q_j0p6[1])
                 *xx+q_j0p6[0]) / res;
        /* Beruecksichtigung der Funktionsnullstellen   */
        /* n1 = q_n1a + q_n1b und n2 = q_n2a + q_n2b    */
        res = ( (x-q_n1a) - q_n1b ) * ( (x-q_n2a) - q_n2b ) * res;
        ...
      else ...                   /* x>6, weitere Teilintervalle */

  }
  ...

#ifdef PXSCTRAP
  E_SPOPP("q_j0");
#endif

return(res);
}
```

Anhang C

Einige spezielle Funktionen

Zur Erinnerung und einfacheren Handhabung wird im folgenden eine Auswahl von häufiger verwendeten speziellen Funktionen der mathematischen Physik aufgeführt (sofern nicht anders angegeben, gilt $x \in \mathbb{R}$, $w, z \in \mathbb{C}$):

- Gammafunktion

$$\Gamma(z) := \int_0^\infty t^{z-1} e^{-t} dt, \quad \text{Re } z > 0$$

- Unvollständige Gammafunktion

$$P(a, x) := \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt \quad x \in \mathbb{R}, a > 0$$

- Betafunktion

$$B(z, w) = B(w, z) := \int_0^1 t^{z-1} (1-t)^{w-1} dt = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)},$$

$$\text{Re } z > 0, \text{Re } w > 0$$

- Fehlerfunktion

$$\text{erf}(z) := \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$$

$$\text{erfc}(z) \equiv 1 - \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-t^2} dt$$

- Besselfunktionen

$$J_v(z) := \left(\frac{1}{2}\right)^v \sum_{k=0}^{\infty} \frac{(-\frac{1}{4}z^2)^k}{k!\Gamma(v+k+1)}, \quad v \in \mathbb{R}$$

$$Y_v(z) := \frac{J_v(z)\cos(v\pi) - J_{-v}(z)}{\sin(v\pi)}, \quad v \in \mathbb{R}\backslash\mathbb{Z}$$

- Modifizierte Besselfunktionen (ganzzahlige Ordnung)

$$I_n(x) := i^{-n} J_n(ix)$$

$$K_n(x) := \frac{\pi}{2} i^{n+1} \left(J_n(ix) + iY_n(ix)\right)$$

- Bessel-Ricatti Funktionen

$$\psi_n(x) := \sqrt{\frac{\pi x}{2}} J_{n+1/2}(x)$$

$$\chi_n(x) := \sqrt{\frac{\pi x}{2}} Y_{n+1/2}(x)$$

- Kelvinsche Funktionen

$$\mathrm{ber}_v z + i\ \mathrm{bei}_v z := J_v(i^{-3/2}z)$$

$$\mathrm{ker}_v z + i\ \mathrm{kei}_v z := i^{\mp v} K_v(\pm i^{1/2} z)$$

- Airyfunktionen

$$\mathrm{Ai}(z) = \frac{1}{\pi}\sqrt{\frac{z}{3}} K_{1/3}(\xi)$$

$$\mathrm{Bi}(z) = \sqrt{\frac{z}{3}}(I_{-1/3}(\xi) + I_{1/3}(\xi))$$

$$\text{mit } \xi := \frac{2}{3} z^{3/2}$$

- Elliptische Integrale

$$F(k,\varphi) := \int_0^{\varphi} \frac{d\varphi}{\sqrt{1 - k^2 \sin^2\varphi}}, \quad 0 \le k \le 1,\ x = \sin\varphi$$

$$E(k,\varphi) := \int_0^{\varphi} \sqrt{1 - k^2 \sin^2\varphi}\, d\varphi, \quad 0 \le k \le 1,\ x = \sin\varphi$$

$$K := F(k, \frac{\pi}{2}), \quad E := E(k, \frac{\pi}{2})$$

- Dawson's Integral
$$F(x) = e^{-x^2} \int_0^x e^{t^2} dt$$
- Exponentielles Integral
$$E_1(x) := \int_z^\infty \frac{e^{-t}}{t} dt$$
- Integralsinus
$$\mathrm{Si}(x) := \int_0^x \frac{\sin t}{t} dt$$
- Integralcosinus
$$\mathrm{Ci}(x) := -\int_x^\infty \frac{\cos t}{t} dt$$
- Fresnel-Integrale
$$C(x) := \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\cos t}{\sqrt{t}} dt$$
$$S(x) := \frac{1}{\sqrt{2\pi}} \int_0^x \frac{\sin t}{\sqrt{t}} dt$$

Anhang D

Friedrich Wilhelm Bessel

Die folgenden biographischen Textauszüge stammen von Sippel [108] und geben einen kleinen Einblick in das Leben und Wirken von Friedrich Wilhelm Bessel wieder:

„Friedrich Wilhelm Bessel wurde am 22.6.1784 in Minden als Sohn von Karl Friedrich und Ernestine Bessel geboren. Wegen der Größe der Familie war die finanzielle Lage immer recht gespannt, doch konnten alle Kinder eine solide Schulausbildung machen.

Friedrich Wilhelm verließ nach der 8. Klasse das Gymnasium und begann wegen seines Interesses an der Mathematik 1799 eine Kaufmannslehre in Bremen. In diese Zeit fallen auch seine ersten Beobachtungen des gestirnten Himmels, bei denen er einen Doppelstern in der Leier entdeckte.

Sein Interesse an der Seefahrt (in erster Linie für Handelszwecke) führte zu einer Beschäftigung mit nautischer Astronomie, für deren Verständnis er sich die Grundlagen der Astronomie und die dazugehörige Mathematik erarbeiten mußte. Die ausführlichen Studien und auch erste systematische Himmelsbeobachtungen waren neben seiner regulären Arbeit nur möglich, da Bessel mit 5 Stunden Schlaf auskam.

Im Jahre 1804 berechnet er aufgrund von neu vorliegenden Daten die Bahn des Halleyschen Kometen und legt seine Arbeit dem „Kometenastronomen“ Heinrich Wilhelm Olbers vor, der Bessels Talent erkennt und sein Förderer und väterlicher Freund wird. (...)

1809 folgt er einem Ruf des preußischen Innenminister nach Königsberg, wo er eine neue Sternwarte aufbauen kann, die er mit erstklassigen Instrumenten ausstattet. 1812, ein Jahr vor der Fertigstellung und seinem Einzug in die Sternwarte heiratet Bessel die Medizinalratstochter Johanna Hagen.

Seinen ersten Sohn nennt er 1814 zu Ehren Olbers' Wilhelm.

Zu den vielen bedeutenden Werken Bessels gehört die 1807 begonnene Arbeit zur Erstellung eines Fundamentalkataloges, also eine Auflistung von Sternen mit ihren absoluten Positionen. Dazu benutzte er die Aufzeichnungen von James Bradley, der 1750-1762 die scheinbaren Positionen von fast 3300 Fixsternen bestimmt hatte.

Bessel machte sich daran alle Fehler von Präzession, Refraktion, Nutation und Aberration zu bestimmen, außerdem untersuchte er Bradleys optisches Instrument selber und berechnete letztendlich für alle Sterne ihre wahre Position, einen mit den damaligen Kenntnissen ganz enorme Arbeit, die für 6 Jahre einen Großteil seiner Zeit in Anspruch nahm, ihm aber auch einige Auszeichnungen einbrachte, wie beispielsweise 1811 den Lalande-Preis für seine in diesem Zusammenhang erstellte Refraktionstafel und 1812 bzw. 14 die auswärtige Mitgliedschaft in der Berliner und der Petersburger Akademie der Wissenschaften. (...)

Friedrich Wilhelm Bessel war aber nicht nur als Astronom tätig, er fand auch noch Zeit für umfangreiche Arbeiten auf anderen Gebieten. So beschäftigte er sich z. B. mit Untersuchungen zur Länge des Sekundenpendels, erstellte das Urmaß des preußischen Fußes (auf 0,00142 mm genau!) und führte zudem eine Bestimmung des Erdradius am Äquator durch, die den bemerkenswert genauen Wert von 6377,4 km lieferte. In den Jahren 1830-37 beschäftigte er sich außerdem intensiv mit der Landvermessung und führte u.a. die „Ostpreußische Gradmessung“ durch, die Fehler auf der Karte Europas korrigierte.

Eine seiner letzten großen astronomischen Arbeiten war die Entdeckung und Berechnung der Parallaxe von 61 Cygni 1838/40. Diese erste verläßliche Parallaxenbestimmung glückte Bessel, da ihm zum einen ein neues exzellentes optisches Gerät aus der Werkstatt Fraunhofers zur Verfügung stand und er zum zweiten von der richtigen Annahme ausging, da die Sterne mit der größten Eigenbewegung die nähesten seien und nicht die hellsten, wie viele seiner Kollegen dachten.

Am 8. April 1846 verstarb Friedrich Wilhelm Bessel nach mehrjährigem Krebsleiden ruhig und friedlich. Begraben wurde auf dem an die Sternwarte angrenzenden Friedhof.“

Literaturverzeichnis

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

[2] Alefeld, G. und Herzberger, J.: *Einführung in die Intervallrechnung.* Bibliographisches Institut, Mannheim (1974).

[3] Alefeld, G. und Herzberger, J.: *Introduction to Interval Computations.* Academic Press, New York (1983).

[4] Amos, D. E., Daniel, S. L., Weston, M. K.: *CDC 6600 Subroutines IBESS und JBESS for Bessel Functions $I_v(x)$ and $J_v(x)$, $x \geq 0$, $v \geq 0$.* ACM Transactions on Mathematical Software, Vol. 3, No. 1, p. 76-92 (1977).

[5] Amos, D. E., Daniel, S. L., Weston, M. K.: *Algorithm 511: CDC 6600 Subroutines IBESS und JBESS for Bessel Functions $I_v(x)$ and $J_v(x)$, $x \geq 0$, $v \geq 0$.* ACM Transactions on Mathematical Software, Vol. 3, No. 1, p. 93-95 (1977).

[6] Amos, D. E.: *Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.* ACM Transactions on Mathematical Software, Vol. 12, No. 3, p. 265-273 (1986).

[7] Amos, D. E.: *Remark on Algorithm 644.* ACM Transactions on Mathematical Software, Vol. 16, No. 4, p. 404 (1990).

[8] Amos, D. E.: *A Remark on Algorithm 644: A Portable Package for Bessel Functions of a Complex Argument and Nonnegative Order.* ACM Transactions on Mathematical Software, Vol. 21, No. 4, p. 388-393 (1995).

[9] Andrews, L. C.: *Special functions for engineers and applied mathematicians.* Macmillan Publishing Company, New York (1985).

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

[11] Bantle, A.: *Ein Kalkül für verläßliche rechnergestützte Fehlerabschätzungen und dessen Anwendung.* Diplomarbeit, Universität Karlsruhe (1998).

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

[13] Barth, B.; Krämer, W.: *Computation of Interval Bounds for Weierstrass' Elliptic Function*, Computing Suppl. 9, pp. 147–159, Springer Verlag (1993).

[14] Bartholomew-Biggs, M. C., Zakovic, S.: *Using Markov's interval arithmetic to evaluate Bessel-Ricatti functions.* Numerical Algorithms 10, p. 261-287 (1995).

[15] Berg, L.: *Asymptotische Darstellungen und Entwicklungen.* VEB Deutscher Verlag der Wissenschaften, Berlin (1968).

[16] Blair, J. M.: *Rational Chebyshev Approximations for the Modified Bessel Functions $I_0(x)$ and $I_1(x)$.* Mathematics of Computation, Vol. 28, No. 126, p. 581-583 (1974).

[17] Blomquist, F.: *Implementierung und Fehlerabschätzungen von PASCAL-XSC Standardfunktionen für ein dezimales Datenformat.* Universität Karlsruhe (1992).

[18] Blomquist, F. und Krämer, W.: *Algorithmen mit garantierten Fehlerschranken für die Fehler- und die komplementäre Fehlerfunktion.* Preprint 97/3 des IWRMM, Universität Karlsruhe (1997). `FTP://iamk4515.mathematik.uni-karlsruhe.de` im Verzeichnis `/pub/iwrmm/preprints`

[19] Bohlender, D., Ullrich, C.: *Standards zur Computerarithmetik.* In *Wissenschaftliches Rechnen*, Herzberg, J. (Herausgeber), Akademie Verlag, Berlin (1995).

[20] Boisvert, D. E., Saunders, B. V.: *Portable Vectorized Software for Bessel Function Evaluation.* ACM Transactions on Mathematical Software, Vol. 18, No. 4, p. 456-469 (1992).
auch erhältlich im Internet, WWW-Seite
(`http://performance.netlib.org/vfnlib/paper.ps`).

[21] Bowman, F.: *Introduction to Bessel Functions.* Dover Publications Inc., New York (1958).

[22] Braune, K., Krämer, W.: *High-Accuracy Standard Functions for Intervals.* In M. Ruschitzka (Ed.): *Computer Systems: Performance and Simulation.* Elsevier Science Publishers (1985).

[23] Braune, K., Krämer, W.: *High Accuracy Standard Functions for Real and Complex Intervals.* In Kaucher, E., Kulisch, U., Ullrich, Ch: *Computerarithmetic: Scientific Computation and Programming Languages*, pp81–114, Teubner, Stuttgart (1987).

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

[25] Campbell, J. B.: *On Temme's Algorithm for the Modified Bessel Function of the Third Kind.* ACM Transactions on Mathematical Software, Vol. 6, No. 4, p. 581-586 (1980).

[26] Cody, W. J.: *The FUNPACK Package of Special Function Subroutines.* ACM Transactions on Mathematical Software, Vol. 1, No. 1, p. 13-25 (1975).

[27] Cody, W. J., Motley, R. M., Fullerton, L. W.: *The Computation of Real Fractional Order Bessel Functions of the Second Kind.* ACM Transactions on Mathematical Software, Vol. 3, No. 3, p. 232-239 (1977).

[28] Cody, W. J., Waite, W.: *Software Manual for the Elementary Functions.* Prentice-Hall, New Jersey (1980).

[29] Cody, W. J.: *Algorithm 597: Sequence of Modified Bessel Functions of the First Kind.* ACM Transactions on Mathematical Software, Vol. 9, No. 2, p. 242-245 (1983).

[30] Cody, W. J., Stoltz, L.: *Performance Evaluation of Programs for Certain Bessel Functions.* ACM Transactions on Mathematical Software, Vol. 15, No. 1, p. 41-48 (1989).

[31] Cody, W. J., Stoltz, L.: *The Use of Taylor Series to Test Accuracy of Function Programs.* ACM Transactions on Mathematical Software, Vol. 17, No. 1, p. 55-63 (1991).

[32] Cordes, D.: *PASCAL-XSC Runtime Library.* SCAN 90 (1990).

[33] Cordes, D., Krämer, W.: *PASCAL-XSC Module for Multiple-Precision Operations and Functions.* Universität Karlsruhe (1991).

[34] Cody, W. J.: *Algorithm 715: SPECFUN – A Portable FORTRAN Package of Special Function Routines and Test Drivers.* ACM Transactions on Mathematical Software, Vol. 19, No. 1, p. 22-32 (1993).

[35] Dekker, T.J.: *Floating-Point Technique for Extending the Availible Precision.* Numerische Mathematik 18, S. 224-242 (1971).

[36] Deuflhard, P.; Hohmnn, A.: *Numerische Mathematik I – Eine algorithmisch orientierte Einführung.* de Gruyter, Berlin (1993).

[37] Ferguson, W. E. Jr. *Exact computation of a sum or difference with applications to argument reduction.* In *Proc. 12th IEEE Symposium on Computer Arithmetic, Bath, England*, Simon Knowles and William H. McAllister, editors, IEEE Computer Society Press, Los Alamitos, CA, USA, pp. 216-221 (1995).

[38] Fischer, H.-C.: *Schnelle automatische Differentiation, Einschliessungsmethoden und Anwendungen.* Dissertation, Universität Karlsruhe (1990).

[39] Gautschi, W.: *Algorithm 236: Bessel Functions of the First Kind.* ACM TOMS, Vol. 7, No. 8, p. 479-480 (1964).

[40] Ghanem, R. B.: *Spherical Bessel Functions and Explicit Quadrature Formula.* Mathematics of Computation, Vol. 66, No. 217, p. 289-296 (1997).

[41] Hammer, R.: *Multi-Precision Arithmetic in PASCAL–XSC, Implementation and Applications.* Talk at SCAN-93, Wien (1993).

[42] Hansen, E.: *Global Optimization Using Interval Analysis – The Multi-Dimensional Case.* Numerische Mathematik **34**, 247–270 (1980).

[43] Hart, J. F. et al.: *Computer Approximations.* Wiley, New York / London / Sydney (1968).

[44] Heuser, H.: *Gewöhnliche Differentialgleichungen – Einführung in Lehre und Gebrauch.* B. G. Teubner, Stuttgart (1989).

[45] Hill, G. W.: *Algorithm 518: Incomplete Bessel Function I_0: The Von Mises Distribution.* ACM Transactions on Mathematical Software, Vol. 3, No. 3, p. 279-284 (1977).

[46] Hill, G. W.: *Evaluation and Inversion of the Ratios of Modified Bessel Functions, $I_1(x)/I_0(x)$ and $I_{1.5}(x)/I_{0.5}(x)$.* ACM Transactions on Mathematical Software, Vol. 7, No. 2, p. 199-208 (1981).

[47] Hill, G. W.: *Algorithm 571: Statistics for von Mises' and Fisher's Distributions of Directions: $I_1(x)/I_0(x)$ and $I_{1.5}(x)/I_{0.5}(x)$ and Their*

Inverses. ACM Transactions on Mathematical Software, Vol. 7, No. 2, p. 233-238 (1981).

[48] Hofschuster, W.: *Vergleich verschiedener Langzahlarithmetiken und Implementierung verschiedener Verfahren zur Multiplikation langer Zahlen.* Diplomarbeit, Universität Karlsruhe (1992).

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

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

[51] Hofschuster, W., Krämer, W.: *A Fast Public Domain Interval Library in ANSI-C*, Proceedings zur IMACS'97 in Berlin, Volume 2, pp. 395-400 (1997).

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

[53] Hofschuster, W., Krämer, W.: *Mathematical Function Software on the Web – Are Such Codes Useful for Verification Algorithms?.* erscheint in Reliable Computing, Issue 2, Volume 6 (2000).

[54] Kaucher, E.: *Über metrische und algebraische Eigenschaften einiger beim numerischen Rechnen auftretender Räume.* Dissertation, Universität Karlsruhe (1973).

[55] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., Ullrich, Ch.: *PASCAL–XSC — Sprachbeschreibung mit Beispielen.* Springer-Verlag, Berlin/Heidelberg/New York (1991).

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

[57] Klatte R., et. al.: *C-XSC, A C++ Class Library for Scientific Computing*, Springer (1993).

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

[59] Krämer, W.: *Mehrfachgenaue reelle und intervallmäßige Staggered-Correction Arithmetik mit zugehörigen Standardfunktionen*, Bericht des Instituts für Angewandte Mathematik, Universität Karlsruhe, S. 1-80 (1988).

[60] Krämer, W.: *Berechnung der Gammafunktion für reelle Punkt- und Intervallargumente*, Z. angew. Math. Mech. 70, pp. 581–584 (1990).

[61] Krämer, W.: *Computation of Verified Bounds for Elliptic Integrals*, Proceedings of the International Symposium on Computer Arithmetic and Scientific Computation, SCAN91, Oldenburg (1991). Edited by J.Herzberger and L. Atanassova; Elsevier Science Publishers (North-Holland).

[62] Krämer, W.: *PASCAL-XSC Modules for Multiple-Precision Interval Operations and Functions*. Universität Karlsruhe (1991).

[63] Krämer, W.: *Die Berechnung von Standardfunktionen in Rechenanlagen*. In Chatterji, S. D., Kulisch, U., Laugwitz, D., Liedl, R., Purkert, W. (Eds.): *Jahrbuch Überblicke Mathematik 1992*, Vieweg, Braunschweig (1992).

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

[65] Krämer, W.: *Eine portable Langzahl- und Langzahlintervallarithmetik mit Anwendungen*. ZAMM **73** (1992).

[66] Krämer, W.: *Die Berechnung von Funktionen und Konstanten in Rechenanlagen*. Habilitationsschrift, Universität Karlsruhe (1993).

[67] Krämer, W.: *Multiple-Precision Computations with Result Verification*. In Adams, E.; Kulisch, U.(eds.): *Scientific Computing with Automatic Result Verification*. I. Language and Programming Support for Verified Scientific Computation, II. Enclosure Methods and Algorithms with Automatic Result Verification, III. Applications in the Engineering Sciences. Academic Press, San Diego, pp 325–356 (1993).

[68] Krämer, W.: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen*, Bericht des Instituts für Angewandte Mathematik, Universität Karlsruhe (1996).

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

[70] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II*. Erscheint demnächst im Springer Verlag.

[71] Kulisch, U.: *Grundlagen des Numerischen Rechnens – Mathematische Begründung der Rechnerarithmetik.* Reihe Informatik, Band 19, Bibliographisches Institut, Mannheim (1976).

[72] Kulisch, U. und Miranker, W. L.: *Computer Arithmetic in Theory and Practice.* Academic Press, New York (1981).

[73] Kulisch, U. und Miranker, W. L.: *The Arithmetic of the Digital Computer: A New Approach.* SIAM Review **28**, No. 1, 1–140 (1986).

[74] Kulisch, U. (Ed.): *Wissenschaftliches Rechnen mit Ergebnisverifikation — Eine Einführung.* Ausgearbeitet von S. Geörg, R. Hammer und D. Ratz. Vol. 58. Akademie Verlag, Berlin, und Vieweg Verlagsgesellschaft, Wiesbaden (1989).

[75] Lebedew, N.N.: *Spezielle Funktionen und ihre Anwendung*, BI-Verlag, Mannheim, Wien, Zürich (1973).

[76] Linnainmaa, F.: *Software for Double-Precision Floating-Point Computations.* ACM Trans. on Math. Software, Vol. **7**, No. 3, pp 272-282 (1981).

[77] Lohner, R.: *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*, Dissertation, Universität Karlsruhe (1988).

[78] Lohner, R.: *Interval Arithmetic in Staggered Correction Format*, in: *Scientific Computing with Automatic Result Verification*, Adams, E., Kulisch, U.(editors), Academic Press, pp. 301–321 (1992).

[79] Lohner, R.: *Verified computing and programs in PASCAL-XSC*, Habilitationsschrift, Universität Karlsruhe (1994).

[80] Lohner, R.: *Verified Computation of Bounds for Special Functions by Use of Generating Functions*, persönliche Mitteilungen (1999).

[81] Lozier, D. W., Olver, F. W.: *Numerical Evaluation of Special Functions.* Proceedings of Symposia in Applied Mathematics, Volume 48, p. 79-125 (1994).

[82] Lozier, D. W.: *Software needs in special functions.* Journal of Computational and Applied Mathematics 66, p. 345-358 (1996).

[83] Luke, Y. L.: *The special funktions and their approximations.* Academic Press, New York, London (1969).

[84] Luke, Y. L.: *Miniaturized Tables of Bessel Functions.* Mathematics of Computation, Vol. 25, No. 114, p. 323-330 (1971).

[85] Luke, Y. L.: *Miniaturized Tables of Bessel Functions II.* Mathematics of Computation, Vol. 25, No. 116, p. 789-795 (1971).

[86] Luther, W., Otten, W.: *Reliable computation of elliptic functions*, Journal of Universal Computer Science (J.UCS), Vol 4, No. 1, 25-33 (1998).
(http://www.iicm.edu/jucs_4_1/reliable_computation_of_elliptic)

[87] MacLeod, A. J.: *Table-based tests for Bessel function software.* Advances in Computational Mathematics 2, p. 251-260 (1994).

[88] MacLeod, A. J.: *Algorithm 757: MISCFUN, A Software Package to Compute Uncommon Special Functions.* ACM Transactions on Mathematical Software, Vol. 22, No. 3, p. 288-301 (1996).

[89] Markov, S. M.: *Some Applications of Extended Interval Arithmetic to Interval Iterations.* Computing, Suppl. 2, p. 69-84 (1980).

[90] Mayer, G.: *Grundbegriffe der Intervallrechnung.* In [74], 101–118 (1989).

[91] McLachlan, N. W.: *Bessel Functions for Engineers.* Oxford University Press, London, Second Edition (1955).

[92] Miller, J. C. P.: *Bessel Funktions, Part II (Math. Tables X).* Cambridge University Press (1952).

[93] Moore, R. E.: *Interval Analysis.* Prentice Hall, Engelwood Cliffs, New Jersey (1966).

[94] Moore, R. E.: *Methods and Applications of Interval Analysis.* SIAM, Philadelphia, Pensylvania (1979).

[95] Moshier, S. L. B.: *Methods and programs for mathematical functions.* Ellis Horwood Limited, Chichester (1989).

[96] Muller, J. M.: *Elementary functions: algorithms and implementation.* Birkhäuser, Boston (1997).

[97] Olver, F. W. J.: *Error Bounds for Linear Recurrence Relations.* Mathematics of Computation, Vol. 50, No. 182, p. 481-499 (1988).

[98] Ooura: *Ooura's Mathematical Software Packages*,
Internet, WWW-Seite
`http://momonga.t.u-tokyo.ac.jp/~ooura/bessel.html`.

[99] Press, W. H., Flannery, B. P., Teukolsky, S. A., Vetterling, W. T.: *Numerical Recipes in Pascal – The Art of Scientific Computing.* Cambridge University Press (1989).

[100] Priest, D. M.: *On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations.* Ph.D. thesis, Mathematics Department, University of California, Berkeley, CA, USA (1992).
`ftp://ftp.icsi.berkeley.edu/pub/theory/priest-thesis.ps.Z`.

[101] Rall, L. B.: *Automatic Differentiation: Techniques and Applications.* Lecture Notes in Computer Science, No. **120**, Springer-Verlag, Berlin (1981).

[102] Ratschek, H. und Rokne, J.: *Computer Methods for the Range of Functions.* Ellis Horwood Limited (1984).

[103] Ratz, D.: *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen.* Dissertation, Universität Karlsruhe (1992).

[104] Ratz, D.: *Inclusion Isotone Extended Interval Arithmetic*, Bericht 5/1996 des Instituts für Angewandte Mathematik, Universität Karlsruhe (1996).

[105] Rehwald, W.: *Elementare Einführung in die Bessel-, Neumann- und Hankel-Funktionen.* Hirzel Verlag, Stuttgart (1959).

[106] Schäfke, F. W.: *Einführung in die Theorie der Spezielle Funktionen der Mathematischen Physik.* Springer-Verlag, Berlin (1963).

[107] Sneddon, I. N.: *Spezielle Funktionen der Mathematischen Physik und Chemie.* BI-Verlag, Mannheim (1963).

[108] Sippel, C. M.: *Wer war Friedrich Wilhelm Bessel?*, TITAN, Zeitschrift für Hobbyastronomen, Ausgabe 1/96 (1996).

[109] Skovgaard, O.: *Remark on Algorithm 236.* ACM TOMS, Vol. 1, No. 3, p. 282-284 (1975).

[110] Spanier, J., Oldham, K. B.: *An Atlas of Functions.* Hemisphere Publishing Corporation, Washington (Springer Verlag)

[111] Srivastava, H. M.; Manocha, H. L.: *A Treatise on Generating Functions.* Ellis Horwood Ltd., Chichester (1984).

[112] Sterbenz, P. H.: *Floating-Point Computation.* Prentice-Hall, Englewood Cliffs, New Jersey, USA (1974).

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

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

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

[116] Temme, N. M.: *On the Numerical Evaluation of the Modified Bessel Function of the Third Kind.* Journal of Computational Physics 19, p. 324-337 (1975).

[117] Thompson, W. J.: *Atlas for Computing Mathematical Functions.* John Wiley & Sohns, New York (1997).

[118] Vrahatis, M. N., Grapsa, T. N., Ragos, O., Zafiropoulos, F. A.: *On the Localization and Computation of Zeros of Bessel Functions.* ZAMM 77, S. 467-475 (1997).

[119] Tranter, C. J.: *Bessel functions with some physical applications.* English Universities Press, London (1968).

[120] Walter, W.: *Gewöhnliche Differentialgleichungen.* Springer Verlag, Berlin, Heidelberg, New York, 6. Aufl. (1996).

[121] Watson, G.N.: *A Treatise on the Theory of Bessel Functions.* Cambridge University Press (1966).

[122] Werner, K.: *Calculation of the inverse Weierstraß Function in an Arbitrary Machine Arithmetic.* In: Scientific Computing and Validated Numerics, Proceedings of SCAN-95, G. Alefeld, A. Frommer, B. Lang eds., Akademie Verlag, Berlin (1996).

[123] Wilkinson, J.H.: *Rundungsfehler.* Springer Verlag, Berlin, Heidelberg (1969).

[124] Zhang, S., Jianming, J.: *Computation of Special Functions.* John Wiley & Sohns, New York (1996).

Lebenslauf

Werner Kurt Hofschuster

geboren am	11. Juni 1968 in Heidenheim/Brenz
Eltern	Karl Hofschuster und Anna Hofschuster, geb. Demel
Schulbildung	1974 – 1975 Grundschule in Schw. Gmünd–Herlikofen 1975 – 1978 Grundschule in Böbingen/Rems 1978 – 1987 Gymnasium in Heubach
Reifeprüfung	21. Mai 1987
Studium	der Wirtschaftsmathematik an der Universität Karlsruhe vom Wintersemester 1987/88 bis Juli 1993
Diplomprüfung	im Studiengang Wirtschaftsmathematik am 22. Juli 1993
Zivildienst	Dezember 1993 bis Februar 1995
Berufstätigkeit	• Januar 1992 bis November 1993 als wissenschaftliche Hilfskraft am Institut für Angewandte Mathematik der Universität Karlsruhe • März 1994 bis Februar 1995 als Nebentätigkeit: Auftrag des Instituts für Angewandte Mathematik über Softwareentwicklungsarbeiten (Standardfunktionen für das binäre IEEE–Format) • März 1995 bis März 2000 als wissenschaftlicher Mitarbeiter am Institut für Angewandte Mathematik und am Institut für Wissenschaftliches Rechnen und Mathematische Modellbildung (IWRMM) der Universität Karlsruhe • seit April 2000 wissenschaftlicher Mitarbeiter der Arbeitsgruppe Wissenschaftliches Rechnen/Softwaretechnologie am Fachbereich Mathematik der Bergischen Universität GH Wuppertal
Eheschließung	am 9.9.1994 mit Brigitte Hofschuster, geb. Kreb
Kinder	Johannes Thomas Hofschuster, geb. am 5.9.1995 Clemens Stefan Hofschuster, geb. am 29.4.1998 Lukas Samuel Hofschuster, geb. am 27.6.2000