Verifizierende Numerik mit PASCAL-XSC BCD-VERSION

STANDARD-FUNKTIONEN IN INTERVAL STAGGERED ARITHMETIC

Frithjof Blomquist

STANDARD-FUNKTIONEN IN INTERVAL STAGGERED ARITHMETIC

66346 Püttlingen 3
E-Mail Adresse: Blomquist@math.uni-wuppertal.de

Inhaltsverzeichnis

Kapitel 1

Standardfunktionen mit einer Variablen

In diesem Kapitel werden die Algorithmen vorgestellt, mit denen die im Buch

Verifizierende Numerik mit
PASCAL-XSC
HILFS-MODULE
BIBLIOTHEK1

angegebenen Standardfunktionen in *interval staggered arithmetic* realisiert wurden. Wir beschränken uns dabei auf den Basistyp `real`, da die Algorithmen zum Basistyp `Longreal` bis auf die entsprechenden internen Konstanten identisch sind.

Die Deklaration

```
var x : Stagg[-1..Stagg_prec];
```

realisiert das folgende Intervall:

$$\mathtt{x} = \mathtt{x}[1] + \mathtt{x}[2] + \ldots + \mathtt{x}[\mathtt{Stagg\_prec}] + [\,\mathtt{x}[-1], \mathtt{x}[0]\,]$$

Die `integer`-Größe `Stagg_prec` aus dem Modul `type_def` legt die Präzision der *interval staggered arithmetic* fest und realisiert so eine Intervallrechnung mit mindestens $(\mathtt{Stagg\_prec} + 1) \cdot 13$ Dezimalstellen.

Eine häufige Anwendung der *interval staggered arithmetic* ist die Auswertung eines gegebenen Polynoms, z.B.:

$$P_4(t) := 1 + \frac{1}{2}t^1 + \frac{1}{3}t^2 + \frac{1}{4}t^3 + \frac{1}{5}t^4, \quad t \in I\!R$$

nach dem Hornerschema über einem gegebenen, reellen Intervall $\mathbf{x} \ni t$, wobei die Polynomkoeffizienten durch staggered Intervalle einzuschließen sind. Das nachfolgende Programm `beispiel` liefert zu einem Eingabeintervall `x : Stagg[-1..Stagg_prec]` mit y eine garantierte Einschließung von $P_4(\mathbf{x}) \subset \mathbf{y}$.

```
program beispiel1;
{ Hornerschema in interval staggered arithmetic:      }
use type_def, { exportiert STagg_prec : integer;      }
    stagg;    { exportiert staggered Operatoren *,+,/ }
var p : integer;
procedure main;
var a  : array[0..4] of Stagg[-1..Stagg_prec];
   x,y : Stagg[-1..Stagg_prec];
   j   : integer;
begin
   a[0] := 1; { Koeffizienteneinschliessungen: }
   for j := 1 to 4 do a[j] := a[0]/(j+1);
   writeln('x = ?');  read(x);
   y := a[4]; { Hornerschema in staggered arithmetic: }
   for j := 4 downto 0 do y := y*x + a[j];
   writeln('Einschliessung von P_5(x):');   writeln(y);
end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p); { Wahl der Praezision }
   main;
end.
```

Mit den entsprechenden Eingabedaten für das Punktintervall $\mathbf{x} = [1, 1]$ erhält man zur Präzision `Stagg_prec=3` das folgende Ablaufprotokoll:

```
Stagg_prec = ?
3
x = ?
--- 3 Stagg components and 1 interval are needed in this sequence ---
1 0 0 [0,0]
Einschliessung von P_5(x):
 2.483333333333E+000
 3.333333333333E-013
 3.333333333333E-026
[  3.333333333333E-039,  3.333333333334E-039 ]
```

Der Polynomwert $P_4(1) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} = 2.4833\ldots$ wird also mit $4 \cdot 13 = 52$ Dezimalstellen durch das Intervall `y` maximalgenau eingeschlossen. Beachten Sie bitte, daß durch diese Einschließung eine Abschätzung des Polynomauswertefehlers jetzt überflüssig ist und nur dann erforderlich wird, wenn die Auswertung des Hornerschemas punktförmig erfolgt.
Abschließend noch einige Anmerkungen zum **Approximationsfehler**:

Das Polynom $P_n(t)$ sei eine Approximation der Funktion $f : I\!R \to I\!R$ über dem reellen Intervall $[a, b] \ni t$. Der relative Approximationsfehler ist definiert durch:

$$(1.1) \qquad \varepsilon_{app} := \frac{f(t) - P_n(t)}{f(t)}, \qquad f(t) \neq 0 \text{ für } t \in [a, b], \quad \varepsilon_{app} = \varepsilon_{app}(t);$$

Für $t \in [a, b]$ sei ε_{app} abgeschätzt durch: $|\varepsilon_{app}| \leq \varepsilon(app) < \frac{1}{2}$, wobei die zusätzliche Forderung $\varepsilon(app) < \frac{1}{2}$ keine wirkliche Einschränkung bedeutet. Aus (1.1) folgt direkt

$$(1.2) \qquad f(t) = \frac{P_n(t)}{1 - \varepsilon_{app}} = P_n(t) \cdot (1 + \varepsilon_1)$$

und unter der obigen Voraussetzung $\varepsilon(app) < 0.5$ gilt die Abschätzung:

$$(1.3) \qquad |\varepsilon_1| < (1 + 2 \cdot \varepsilon(app)) \cdot \varepsilon(app) =: \varepsilon(1)$$

Für den Funktionswert $f(t)$ erhält man dann nach (1.2) die Einschließsung

$$(1.4) \qquad \langle f(t) \rangle = P_n(t) \cdot [1 - \varepsilon(1), 1 + \varepsilon(1)]$$

Ist nun $\mathtt{x} \subseteq [a, b]$ ein Teilintervall aus $[a, b]$ und bedeutet $\langle P_n(\mathtt{x}) \rangle$ eine Einschließung aller Polynomwerte $P_n(t)$, $t \in \mathtt{x}$, wobei $\langle P_n(\mathtt{x}) \rangle$ z.B. mit dem Intervall–Hornerschema nach Seite 8 berechnet werden kann, so ist für die Funktionswerte $f(\mathtt{x})$ eine Einschließung gegeben durch:

$$(1.5) \qquad \langle f(\mathtt{x}) \rangle = \langle P_n(\mathtt{x}) \rangle \cdot [1 - \varepsilon(1), 1 + \varepsilon(1)]$$

dabei bedeutet das Symbol $\cdot$ in (1.5) die Multiplikation zweier reeller Gleitpunktintervalle. Der als bekannt vorausgesetzte Approximationsfehler $\varepsilon(app)$ kann im Algorithmus also einfach dadurch berücksichtigt werden, daß in (1.5) mit dem Intervall $[1-\varepsilon(1), 1+\varepsilon(1)]$ multipliziert wird, wobei $\varepsilon(1)$ und $\varepsilon(app)$ über (1.3) im Zusammenhang stehen. Die Multiplikation mit dem speziellen Intervall $[1 - \varepsilon(1), 1 + \varepsilon(1)]$ sollte dabei aus Laufzeitgründen mit dem Prozeduraufruf

`mul1pmeps(y,eps1);` $\quad \langle P_n(\mathtt{x}) \rangle = \mathtt{y}$ `: Stagg;` $\quad \varepsilon(1) = \mathtt{eps1}$ `: real`

erfolgen.

1.1 Das Quadrat

Zu x : `Stagg[-1..Stagg_prec]` ist das Quadrat x^2 definiert durch:

$$x^2 := \{t^2 \mid t \in x\}$$

Mit der Anweisung $y := \texttt{sqr}(x)$ gilt dann: $x^2 \subseteq y$, wobei die Funktion `sqr(x)` vom Hilfsmodul `s_fu1` exportiert und durch den folgenden Quelltext definiert wird:

```
function sqr(var x : Stagg) : Stagg[-1..Stagg_prec];
var y : Stagg[-1..Stagg_prec];
begin
   y := abs(x);
   sqr := y * y;
end; { sqr }
```

Beachten Sie bitte, daß sich die Ergebnisse von `sqr(x)` und `x*x` deutlich unterscheiden, wenn die Null im Innern von `x` enthalten ist. Wir betrachten dazu das folgende Beispiel:
Mit `Stagg_prec=3` und `x = 0 + 0 + 0 + [-1,4]` erhält man:

```
sqr(x) =
 8.000000000000E+000
-0.000000000000E+000
-0.000000000000E+000
[ -8.000000000000E+000,  8.000000000000E+000 ]
```

und damit das Intervall $[0, 16]$, während `x*x` das Ergebnis

```
 6.000000000000E+000
-0.000000000000E+000
-0.000000000000E+000
[ -1.000000000000E+001,  1.000000000000E+001 ]
```

liefert, wobei dieses Intervall $[-4, 16]$, auch negative Elemente enthält, wie es bei der Intervallmultiplikation $[-1, 4] \cdot [-1, 4] = [-4, 16]$ auch zu erwarten ist. Bei fast allen praktischen Anwendungen wird jedoch die Null nicht in x enthalten sein, so daß `sqr(x)` und `x*x` fast immer die gleichen Ergebnisse liefern werden. Es gilt jedoch stets: $\texttt{sqr(x)} \subseteq \texttt{x} * \texttt{x}$.

1.2 Die Potenzfunktion $x^n, n = 0, \pm 1, \pm 2, \ldots$

Zu x : `Stagg[-1..Stagg_prec]` und n : `integer` ist die Potenzfunktion x^n definiert durch:

$$x^n := \{t^n \,\big|\, t \in x, n = 0, \pm 1, \pm 2, \ldots\}$$

Mit der Anweisung $y :=$ `power`(x,n) gilt dann: $x^n \subseteq y$, wobei `power(x,n)` vom Hilfsmodul `stagg` exportiert und durch den folgende Quelltext definiert wird:

```
function power(var x : Stagg; n : integer) :
                                        Stagg[-1..Stagg_prec];
{ Berechnet Einschliessung von x^n fuer ganzzahliges n }
{ Bei  y := power(x,n)  muss gelten:       ub(y)=ub(x) }
var r,k  : integer;
   neg : boolean;
   h,res   : Stagg[-1..ub(x)];
begin
   neg := n<0;
   if neg then n := -n;
   if n=0 then res := 1 else
      if n=1 then res := x else
      begin
         h := abs(x);   k := -1;
         while n > 0 do
         begin { n > 1 }
            k := k + 1;
            r := n mod 2;
            if k=0 then
               if r=1 then res := x else res := 1
               else
               begin
                  h := h*h; if r=1 then res := res*h;
               end;
            n := n div 2;
         end;
      end;
   if neg then res := 1/res;
   power := res;
end; { power }
```

Zunächst werden die beiden Fälle $n = 0$ und $n = 1$ gesondert behandelt und negative Exponenten auf den Fall positiver Exponenten zurückgeführt. Im Fall $n > 1$ wird dann x^n nicht durch $(n-1)$ zeitaufwendige Multiplikationen berechnet sonder in der obigen `while`–Schleife mit Hilfe des schnellen Legendre–Algorithmus, der die eindeutige Binärdarstellung einer positiven ganzen Zahl ausnutzt:

$$n = \sum_{i=0}^{r} a_i \cdot 2^i, \quad a_i \in \{0, 1\}$$

Ist n gerade, so erhält man mit `y := power`(x,n) eine Einschließung der Potenz $x^n \subseteq$`y`, wobei das Intervall `y` keine negativen Elemente enthält.

Mit `Stagg_prec = 22`, $x = 1.5$ und $n = 1000$ liefert `y := power`(x,n) für 1.5^{1000} die garantierte Einschließung:

```
y := power(x,n) =
 1.233840596906E+176
 1.734792274390E+163
 9.948678005742E+150
 1.869005148428E+137
 8.542522975184E+123
 3.294226236528E+110
 4.119393224513E+097
 4.021231941529E+084
 4.553081145119E+071
 2.442800027355E+058
 7.831297406949E+045
 8.413511675680E+032
 2.430116410527E+019
 1.903064987355E+006
 4.242577840019E-007
 8.654766571004E-020
 2.912939505908E-033
 6.549691827790E-046
 3.957702240319E-059
 9.709064245325E-073
 7.682438912543E-086
 1.069834066069E-099
[  3.942571104859E-112,  3.942571104863E-112 ]
```

1.3 Die Quadratwurzel

1.3.1 Der Algorithmus

Die Einschließung der Quadratwurzel $\sqrt{x}$ zu einem vorgegebenen staggered Intervall–Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ erfolgt mit Hilfe eines iterativen Newton–Verfahrens [4], dabei ist $\sqrt{x}$ definiert durch:

$$\sqrt{x} := \left\{ y \,|\, y = \sqrt{\zeta},\ \zeta \in x \right\}$$

Realisiert die staggered Variable z die genannte Einschließung, so gilt mit $S :=$ `Stagg_prec`:

$$\zeta \in x \quad \Longrightarrow \quad \sqrt{\zeta} \in \sqrt{x} \subseteq z = \sum_{i=1}^{S} z_i + [z_{-1}, z_0]. \tag{1.6}$$

Die Berechnung der $z_j \equiv z[j]$, $1 \leq j \leq S$, erfolgt nach folgendem Iterationsverfahren[1]:

$$z_1 = \sqrt{\Box\left(\sum_{i=1}^{\mathrm{ub}(x)} x_i\right)}, \qquad z_1 \text{ ist erste } \texttt{real}\text{–Näherung}; \tag{1.7}$$

$$z_{k+1} = \Box\left(\sum_{i=1}^{\mathrm{ub}(x)} x_i - \sum_{i,j=1}^{k} z_i z_j\right) \Big/ (2z_1), \quad k = 1, \ldots, S-1; \tag{1.8}$$

$$[z_{-1}, z_0] = \frac{\Diamond\left(x - \sum\limits_{i,j=1}^{S} z_i z_j\right)}{\sqrt{\Diamond(x)} + \Diamond\left(\sum\limits_{i=1}^{S} z_i\right)} \tag{1.9}$$

Anmerkungen:

- $\Box$(...) rundet den Ausdruck (...) zur nächsten `real`–Zahl.
- Der Operator $\Diamond$(...) liefert ein Intervall vom Typ `interval`, das den Intervallausdruck (...) im `real`–Format optimal einschließt.
- Die in (1.8) nach dem Newton–Verfahren bestimmten z_j überlappen sich nicht, da der in (1.8) geklammerte Defekt der Approximation $\sum_{i=1}^{k} z_i$ auf der Maschine *maximalgenau* ausgewertet wird.

[1] Die `integer`-Größe `Stagg_prec` bestimmt die Präzision der staggered Arithmetik und wird vom Modul `type_def` exportiert. $13 \cdot ($`Stagg_prec`$+ 1)$ ist die Zahl der benutzten Dezimalziffern.

- Im beschriebenen Algorithmus und im Programm–Quelltext kann der `integer`–Wert $\mathrm{ub}(x)$ und der Parameter $S =$ `Stagg_prec` verschieden gewählt werden.

Daß die nach (1.7), (1.8), (1.9) berechnete staggered Variable

$$z = z_1 + z_2 + \ldots + z_S + [z{-1}, z_0]$$

das Intervall $\sqrt{x}$ auch tatsächlich einschließt, ergibt sich aus der für alle $\gamma \in [x_{-1}, x_0]$ geltenden Identität:

$$\sqrt{\sum_{i=1}^{\mathrm{ub}(x)} x_i + \gamma} = \sum_{j=1}^{S} z_j + \frac{\sum\limits_{i=1}^{\mathrm{ub}(x)} x_i + \gamma - \sum\limits_{i,j=1}^{S} z_i z_j}{\sqrt{\sum\limits_{i=1}^{\mathrm{ub}(x)} x_i + \gamma} + \sum\limits_{j=1}^{S} z_j} =: \alpha(\gamma)$$

Zunächst gilt nach (1.6) die Beziehung

$$\sqrt{x} = \{y \,|\, y = \sqrt{\sum_{i=1}^{\mathrm{ub}(x)} x_i + \gamma},\ \gamma \in [x_{-1}, x_0]\} = \{y \,|\, y = \alpha(\gamma),\ \gamma \in [x_{-1}, x_0]\}$$

Ersetzt man nun im zweiten Summanden von $\alpha(\gamma)$ den Wert $\gamma \in [x_{-1}, x_0]$ durch das Intervall $[x_{-1}, x_0]$, so gilt nach der Inklusionsmonotonie [2]

$$\begin{aligned} \sqrt{x} \quad &\subseteq \quad \sum_{j=1}^{S} z_j + \frac{\sum\limits_{i=1}^{\mathrm{ub}(x)} x_i + [x_{-1}, x_0] - \sum\limits_{i,j=1}^{S} z_i z_j}{\sqrt{\sum\limits_{i=1}^{\mathrm{ub}(x)} x_i + [x_{-1}, x_0]} + \sum\limits_{j=1}^{S} z_j} \\ &= \quad \sum_{j=1}^{S} z_j + \frac{x - \sum\limits_{i,j=1}^{S} z_i z_j}{\sqrt{x} + \sum\limits_{j=1}^{S} z_j} \end{aligned} \tag{1.10}$$

Da der Bruch rechts in (1.10) durch das in (1.9) berechnete Intervall $[z_{-1}, z_0]$ eingeschlossen wird, ergibt sich daraus die Behauptung:

$$\sqrt{x} \subseteq z_1 + z_2 + \ldots + z_S + [z_{-1}, z_0]$$

Skalierung:

Der beschriebene Algorithmus verlangt in (1.8) die Division durch $(2 \cdot z_1)$. Bei großen Präzisionsparametern $S =$ `Stagg_prec` kann es dabei zum Underflow kommen, wenn das Intervallargument x zu kleine Komponenten $x[i]$ enthält. Um entsprechende Fehlermeldungen zu vermeiden, wird das Argument x der Wurzelfunktion nach

$$\sqrt{x} = \sqrt{10^p \cdot x} \cdot 10^{-p/2}, \qquad p = 2, 4, \dots$$

mit maximalem p so skaliert, daß $10^p \cdot x$ für keine staggered Komponente $x_j = x[j]$ einen Overflow erzeugt. Ist k der mit `k := expo(x[j])` berechnete maximale Zehnerexponent aller $x[j]$, so ergibt sich der Skalierungsparameter `p` aus den Anweisungen

```
{ Berechnung der Skalierungs-Exponenten   p,r:        }
r := (254-k) div (-2);       { r: zur Rueckskalierung }
p := -2*r;                   { p: zur Startskalierung }
```

des nachfolgenden Programmquelltextes. Beachten Sie bitte, daß obige Skalierung mit dem Faktor 10^p wegen $p > 0$ durch Anwendung des schnellen Operators `M_10HOCH` rundungsfehlerfrei erfolgt. Die Rückskalierung mit dem Faktor 10^r, $r < 0$ muß jedoch durch wirkliche Multiplikation mit Hilfe der Anweisung `z := comp(0.1,r+1) * z` erfolgen, da der Operator `M_10HOCH` im denormalisierten `real`–Bereich nicht mehr rundungsfehlerfrei arbeitet!

1.3.2 Der Programm-Quelltext

```
function sqrt(var x: Stagg): Stagg[-1..Stagg_prec];
{ Square root( Stagg ) }
var lo    : dotprecision;
    i,j   : integer;
    k,p,r : integer;
    z     : Stagg[-1..Stagg_prec];
    xx    : Stagg;
    z1    : real;
    C     : interval;
begin
  if x = 0 then sqrt := 0 else
  begin
     z1 := x[0] -> x[-1]; { z1: absoluter Durchmesser von x }
     j := expo(z1);  i := j;
```

```
 if SIGN(z1) > 0 then
    if SIGN(x[1]) <> 0 then i := j-expo(x[1])
    else if SIGN(x[0]) <> 0 then i := j-expo(x[0])
    else i := j-expo(x[-1]);
      { i:  Zehner-Exponent des rel. Durchmessers von x }
 if i > -13 then
begin { Die folgende Einschliessung der  Wurzel(x)   im }
      { real-Format mit  sqrt(C) ist bei diesem grossen }
      { relativen Intervall-Durchmesser von x sehr viel }
      { enger als die Einschliessung, die mit Hilfe des }
      { staggered Algorithmus im folgenden   else-Block }
      { berechnet wird !                                }
   C := x;
   sqrt := sqrt(C);
end        else { i <= -13 }
begin
  k := expo(0);
  for j := -1 to ub(x) do
  begin
    p := expo(x[j]);
    if p > k then k := p;
  end;        { k:   maximaler Zehnerexponent der x[j] }
  { Berechnung der Skalierungs-Exponenten   p,r:       }
  r := (254-k) div (-2);      { r: zur Rueckskalierung }
  p := -2*r;                  { p: zur Startskalierung }
  allocate(xx,-1..ub(x));
  for j := -1 to ub(x) do xx[j] := x[j];
  { Startskalierung von xx mit dem Zehner-Exponenten p }
  for j := -1 to ub(x) do xx[j] := xx[j] M_10hoch p;
  lo := #( for i:= 1 to UB(x) SUM( xx[i] ) );
  z1 := SQRT( #*(lo) );
  z[1] := z1;
  For i:= 2 to Stagg_prec do
  begin
    lo := #(lo - For j:= 1 to i-2 SUM(z1*z[j] + z1*z[j])
                                            - z1*z1 );
    z1 := #*(lo) / (2*z[1]);
    z[i] := z1
  end;
  lo := #( lo - For j:= 1 to Stagg_prec-1
```

```
                          SUM( z1*z[j] + z1*z[j] ) - z1*z1 );
      z[-1] := 0;
      z[ 0] := 0;
      C := intval( #<(lo+xx[-1]), #>(lo+xx[0]) );
      C := C / ( SQRT(ival(xx)) + ival(z) );
      free(xx);   z[-1] := C.Inf;  z[ 0] := C.Sup;
      z := comp(0.1,r+1) * z;  { Dies ist die Rueckskalierung }
      sqrt := z;
    end;
  end;
end;
```

Im obigen Programm wird z.B. $\sqrt{\Diamond(x)}$ im Nenner von (1.9) eingeschlossen durch den Funktionsaufruf: `SQRT(ival(xx))`.

1.3.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
  procedure main;
  var x,y : Stagg[-1..Stagg_prec];
      n : integer;
  begin
     repeat
        writeln('x: Stagg  = ?');   read(x);
        y := sqrt(x);
        writeln('sqrt(x)  enthalten in: ');
        writeln(y);
     until false;
  end; { main }

 begin
    writeln('Stagg_prec = ?');  read(p);
    set_Stagg_prec(p);
    main;
 end.
```

wurden die nachfolgenden Einschließungen der Quadratwurzel berechnet:

```
Stagg_prec = ?
10     <enter>
x: Stagg  = ?
10 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0 0 0 0 0    <enter>
sqrt(x)  enthalten in:
 3.162277660168E-134
 3.793319988935E-147
 4.443271853371E-160
 9.555139325216E-173
 8.268575048527E-186
 9.259443863923E-199
 8.221344248108E-212
 3.793002951873E-225
 4.728415284005E-238
 5.148548856030E-251
[  4.538000000000E-264,  4.539000000000E-264 ]
```

Das folgende Beispiel zeigt, daß eine größere Präzision keine höhere Genauigkeit liefern kann, da im `real`–Format 10^{-267} die kleinste positive Zahl ist:

```
Stagg_prec = ?
11     <enter>
x: Stagg  = ?
11 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0 0 0 0 0 0    <enter>
sqrt(x)  enthalten in:
 3.162277660168E-134
 3.793319988935E-147
 4.443271853371E-160
 9.555139325216E-173
 8.268575048527E-186
 9.259443863923E-199
 8.221344248108E-212
 3.793002951873E-225
 4.728415284005E-238
 5.148548856030E-251
 4.538000000000E-264
[ -0.000000000000E+000,  1.000000000000E-267 ]
```

```
Stagg_prec = ?
30     <enter>
x: Stagg  = ?
30 Stagg components and 1 interval are needed in this sequence
1e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 [1e-267,3e-267]    <enter>
sqrt(x)  enthalten in:
 3.162277660168E+127
 3.793319988935E+114
 4.443271853371E+101
 9.555139325216E+088
 8.268575048527E+075
 9.259443863923E+062
 8.221344248108E+049
 3.793002951873E+036
 4.728415284005E+023
 5.148548856030E+010
 4.538800146905E-003
 1.959670015390E-016
 3.344921657179E-029
 2.599406591501E-042
 5.347411333948E-055
 4.124085316929E-068
 5.770904715764E-081
 6.104436925787E-094
 9.062037808609E-107
 9.418283717115E-120
 4.840632855299E-133
 9.118596824564E-146
 2.033269616046E-159
 9.131433612894E-172
 9.791890266529E-185
 5.436126761787E-198
 8.135006138818E-211
 6.278580463683E-224
 1.349524780311E-237
 4.376933467197E-250
[  3.819000000000E-263,  3.820000000000E-263 ]
```

1.4 $\sqrt{x+1}-1$

1.4.1 Der Algorithmus

Ist $\sqrt{x+1}-1 := \{t \,|\, t = \sqrt{\zeta+1}-1,\ \zeta \in x\}$ für das staggered Intervall $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ durch einfache Auswertung des Ausdrucks `sqrt(x+1)-1` einzuschließen, so erhält man umso größere Intervallüberschätzungen, je mehr sich das i.a. sehr schmale Intervall x dem Ursprung nähert. Um die störenden Auslöschungseffekte[2] zu vermeiden, benutzt man zur Approximation von $\sqrt{x+1}-1$ ein Taylorpolynom $P_N(x)$ mit einem umso größerern Polynomgrad N, je größer die geforderte Genauigkeit ist. Mit ε_0 bezeichnen wir die Oberschranke des Betrages des absoluten Approximationsfehlers. Ist y die staggered Einschließung von $\sqrt{x+1}-1 \subseteq y$ auf der Maschine, so lautet der Grobalgorithmus:

if inf$(x) > 1$ then $y :=$ sqrt(x + 1) − 1
else
if $x \subset [-10^{-13}, +10^{-13}]$ then $y := P_N(x) + [-\varepsilon_0, +\varepsilon_0]$
else $y :=$ sqrt(x + 1) − 1; Auswertg. in höherer Präzision.

Da die Auswertung des Ausdrucks `sqrt(x+1)-1` auf der Maschine in einer staggered Intervallarithmetik erfolgt, ergibt sich die geforderte Einschließung $\sqrt{x+1}-1 \subseteq y :=$ sqrt(x + 1) − 1 automatisch. Daher können wir uns jetzt auf das Intervall $[-10^{-13}, +10^{-13}]$ beschränken, in dem $\sqrt{x+1}-1$ mit einem geeigneten Taylorpolynom $P_N(x)$ approximiert wird.

1.4.1.1 Approximation von $\sqrt{x+1}-1$

Im Falle $x \subset [-10^{-13}, +10^{-13}]$ wird $\sqrt{x+1}-1$ approximiert durch das Taylorpolynom $P_N(x)$ vom Grade N:

$$\sqrt{x+1}-1 \approx P_N(x) := \sum_{k=1}^{N} (-1)^k \cdot \left[\prod_{n=0}^{k-1} \frac{2n-1}{2n+2}\right] \cdot x^k, \qquad N = 1, 2, \ldots$$

Mit $\zeta \in x$ gilt für den Betrag des absoluten Fehlers die Abschätzung:

$$\begin{aligned} |\varepsilon_{abs}(\zeta)| &:= \left| \sqrt{\zeta+1} - 1 - P_N(\zeta) \right| \\ &= \left| \sum_{k=N+1}^{\infty} (-1)^k \cdot \left[\prod_{n=0}^{k-1} \frac{2n-1}{2n+2}\right] \cdot \zeta^k \right| \end{aligned}$$

[2]Diese Auslöschungseffekte treten auch bei punktförmigen Intervallen x auf!

$$|\varepsilon_{abs}(\zeta)| \leq \sum_{k=N+1}^{\infty} \left| \prod_{n=0}^{k-1} \frac{2n-1}{2n+2} \right| \cdot |\zeta|^k \leq \frac{1}{2} \cdot \frac{|\zeta|^{N+1}}{1-|\zeta|} \tag{1.11}$$

wobei die Oberschranke des absoluten Fehlers rechts in (1.11) streng monoton in $|\zeta|$ ist. Bedeutet dann ζ_x eine Oberschranke aller Betragselemente aus x, so gilt die Abschätzung

$$|\varepsilon_{abs}(\zeta)| \leq \frac{1}{2} \cdot \frac{\zeta_x^{N+1}}{1-\zeta_x}, \qquad |\zeta| \leq \zeta_x \text{ für alle } \zeta \in x\,; \tag{1.12}$$

Um eine garantierte Oberschranke des absoluten Approximationsfehlers zu erhalten, muß die rechte Seite von (1.12) intervallmäßig ausgewertet werden. Im Programm wird ζ_x durch die `real`–Variable `zx` realisiert und durch die Variable `zi: interval` eingeschlossen. Die Anweisung

$$\texttt{zi} := \texttt{0.5} * \texttt{power(zi,N+1)/(1-zi)} \tag{1.13}$$

liefert dann eine garantierte Einschließung der rechten Seite von (1.12), und der Funktionswert `intval(-zi.sup,+zi.sup)` ist dann eine Einschließung des absoluten Fehlers bei der Approximation $\sqrt{x+1}-1 \approx P_N(x)$, d.h.

$$\sqrt{x+1}-1 \subseteq P_N(x) + \texttt{intval}(-\texttt{zi.sup}, +\texttt{zi.sup}) \tag{1.14}$$

1.4.1.2 Auswertung von $P_N(x)$

Die Auswertung des Approximationspolynoms

$$P_N(x) := \sum_{k=1}^{N} (-1)^k \cdot \left[\prod_{n=0}^{k-1} \frac{2n-1}{2n+2} \right] \cdot x^k, \qquad N = 1, 2, \ldots$$

erfolgt auf dem Rechner durch:

$$P := \frac{3-2N}{2N} \cdot x\,;$$

$$\texttt{For } j := N-1 \texttt{ downto } 1 \texttt{ do } P := (P+1) \cdot \frac{3-2j}{2j} \cdot x\,;$$

Da alle Rechenoperationen in staggered Arithmetik ausgeführt werden, gilt:

$P_N(x) \subseteq P$, P wird im Programm durch die Variable `res` realisiert.

1.4.1.3 Der Polynomgrad N

Jetzt muß noch geklärt werden, wie zu vorgegebenem Präzisionsparameter S := `Stagg_prec` der Polynomgrad N zu wählen ist, damit die durch S geforderte Genauigkeit von $D := 13 \cdot (S+1)$ korrekten Dezimalziffern erreicht wird. Da für $x \subseteq [-10^{-13}, +10^{-13}]$ die Funktionswerte $\sqrt{x+1}-1$ die gleiche Größenordnung wie $x/2$ besitzen und $D = 13 \cdot (S+1)$ korrekte Dezimalziffern zu berechnen sind, muß mit $p := \mathrm{expo}(\zeta_x) - 1 - D$ für den absoluten Approximationsfehler verlangt werden:

$$|\varepsilon_{abs}| < 10^p, \qquad p = \mathrm{expo}(\zeta_x) - 1 - D, \quad D = 13 \cdot (S+1);$$

Da im `real`–Format 10^{-267} die kleinste positive Zahl ist, macht es keinen Sinn, den absoluten Fehler kleiner als 10^{-267} zu wählen. Für p erfolgt daher die anschließende Abfrage: `if p < -267 then p := -267;` Mit (1.12) erhält man dann die Bedingung

$$\frac{1}{2} \cdot \frac{\zeta_x^{N+1}}{1-\zeta_x} < 10^p, \tag{1.15}$$

die vereinfacht werden kann zu:

$$\zeta_x^{N+1} < 10^p \quad \Longleftrightarrow \quad N > \frac{p}{\log_{10} \zeta_x} - 1 \tag{1.16}$$

Im Programm wird die Berechnung von N realisiert durch die Anweisungen:

```
p := expo(zx)-1-13*(Stagg_prec+1);
if p < -267 then p := -267;
N := trunc( p/log10(zx) );
```

wobei die Schranke ζ_x im Programm vorher durch die `real`–Variable `zx` nach oben abgeschätzt wird. Beachten Sie bitte, daß das Weglassen des Faktors 1/2 in (1.15) den Polynomgrad N eher vergrößert und daß in (1.15) der Nenner $(1-\zeta_x)$ durch 1 ersetzt werden kann, ohne die Berechnung von N zu beeinflussen.

Mit dem nach (1.16) berechneten Polynomgrad N wird anschließend im Programm eine Oberschranke des absoluten Approximationsfehlers in (1.13) durch `zi` eingeschlossen, und nach (1.14) gilt dann:

$$\sqrt{x+1} - 1 \subseteq \mathtt{P} + \mathtt{intval}(-\mathtt{zi.sup}, +\mathtt{zi.sup})$$

1.4.2 Der Programm-Quelltext

```
function sqrt1pm1(var x : Stagg) : Stagg[-1..Stagg_prec];
const c  = 1e-13;
      c1 = 1;
var dot   : dotprecision;
   t,res  : Stagg[-1..Stagg_prec];
   t1     : Stagg;
   u,v,zx : real;
   zi     : interval;
   p,N,j  : integer;
begin
   t := x;  normal(t);
   dot := #( t[-1] + 1 + for p:=1 to ub(t) sum(t[p]) );
   if SIGN(dot) < 0 then s_fu1_math_error(8);
   u := t[1] + t[-1];   v := t[1] + t[0];
   if u > c1 then res := sqrt(1+t)-1
   else
      if (u>-c) and (v<c) then
      begin { Approximation durch Taylorpolynom: }
         if (t[1] = 0) and (diam(t)=0) then res := 0
         else
         begin { Berechnung des Polynomgrades N: }
            zi := abs( ival(t) );  zx := zi.sup;
            p := expo(zx)-1-13*(Stagg_prec+1);
            if p < -267 then p := -267;
            N := trunc( p/log10(zx) );
            { Berechnung einer Oberschranke zi.sup }
            { des absoluten Approximationsfehlers: }
            zi := zx; { Intervallauswertung! }
            zi := 0.5 * power(zi,N+1)/(1-zi);
            { Auswertung des Taylor-Polynoms vom Grade N: }
            p := 2*N;  v := 3.0-p;
            u := v /> p;
            if u = v /< p then res := t * u
                          else res := v*t / p;
            for j := N-1 downto 1 do
            begin
               p := 2*j;
```

```
            res := (res+1)*t;
            case j of
              1  : res := res * 0.5;
              2  : res := res * (-0.25);
              3  : res := res * (-0.5);
              4  : res := res * (-0.625);
              5  : res := res * (-0.7);
              6  : res := res * (-0.75);
              8  : res := res * (-0.8125);
              10 : res := res * (-0.85);
              12 : res := res * (-0.875);
              15 : res := res * (-0.9);
              16 : res := res * (-0.90625);
              20 : res := res * (-0.925);
            else : res := res * (3.0-p)/p;
            end;  {case}
          end;
          { Beruecksichtigung des Approximationsfehlers: }
          t := intval(-zi.sup,zi.sup);
          res := res + t;
        end;
      end                 else
      begin
        { Berechnung von    sqrt(x+1)-1    mit der }
        { Praezision: Stagg_prec := Stagg_prec + 2 }
        p := get_Stagg_prec;
        allocate(t1,-1..p+2);
        set_Stagg_prec(p+2);
        t1 := change_length(t);
        t1 := sqrt(t1+1) - 1;
        set_Stagg_prec(p);  { wieder alte Praezision: }
        res := change_length(t1);     free(t1);
      end;
   sqrt1pm1 := res;
end; { sqrt1pm1 }
```

In obiger `case`–Anweisung werden für $j \leq 20$ alle Quotienten $(3 - 2j)/(2j)$ explizit angegeben, deren Werte in einer `real`–Variablen rundungsfehlerfrei gespeichert werden können. Dadurch vermeidet man die sonst notwendige staggered Division durch $(2 \cdot j)$ und reduziert die Laufzeit um etwa 20%.

1.4.3 Numerische Ergebnisse

Das nachfolgende Programm

```
program tests;
{ Einschliessung von sqrt(x+1)-1 }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := sqrt1pm1(x);
       writeln('sqrt1pm1(x)  enthalten in: ');
       writeln(y);
       writeln;
    until false;
 end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für die Einschließung von $\sqrt{x+1}-1$ die folgenden Ergebnisse:

```
Stagg_prec = ?
5      <enter>
x: Stagg   = ?
5 Stagg components and 1 interval are needed in this sequence
5e-13 0 0 0 0 0    <enter>
sqrt1pm1(x)  enthalten in:
 2.499999999999E-013
 6.875000000000E-026
 7.812499999997E-039
 5.585937500008E-052
 5.449218749965E-065
[  2.956542968762E-077,  2.956542968763E-077 ]
```

```
Stagg_prec = ?
20
x: Stagg   = ?
20 Stagg components and 1 interval are needed in this sequence
1e-14 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
sqrt1pm1(x)  enthalten in:
 4.999999999999E-015
 9.875000000000E-028
 6.249999999999E-044
 9.609375000000E-057
 2.734374999999E-072
 9.794921875000E-085
 1.611328124999E-100
 9.869079589843E-113
 7.510910034179E-126
 6.874072647094E-139
 7.265705089569E-152
 9.179617421627E-166
 4.467835429906E-180
 8.450922199338E-193
 6.745453379012E-206
 6.442909195577E-219
 5.162205104175E-232
 6.838094442705E-245
 4.934494663000E-258
 0.000000000000E+000
[ -2.000000000000E-267,  3.000000000000E-267 ]

Stagg_prec = ?
5
x: Stagg   = ?
5 Stagg components and 1 interval are needed in this sequence
1e-30 0 0 0 0 0
sqrt1pm1(x)  enthalten in:
 4.999999999999E-031
 9.999999999999E-044
 9.999875000000E-057
 6.250000000000E-092
-0.000000000000E+000
[ -5.000000000001E-121,  5.000000000001E-121 ]
```

1.5 $\sqrt{1-x^2}$

1.5.1 Der Algorithmus

Mit dem staggered Intervallargument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ gilt zunächst die Definition:

$$\sqrt{1-x^2} := \left\{ t \,|\, t = \sqrt{1-\zeta^2},\ \zeta \in x \subseteq [-1,+1] \right\}$$

Mit der Identität

$$\left\{ t \,|\, t = \sqrt{1-\zeta^2},\ \zeta \in x \right\} \equiv \left\{ t \,|\, t = \sqrt{(1-\zeta)\cdot(1+\zeta)},\ \zeta \in x \right\}$$

folgt dann wegen der Inklusionsmonotonie der Intervallrechnung:

$$\sqrt{1-x^2} \subseteq \sqrt{(1-x)\cdot(1+x)}, \tag{1.17}$$

wobei die Multiplikation rechts in (1.17) als Intervallmultiplikation zu verstehen ist. Für Intervalle x, die nicht zu dicht an den möglichen Randpunkten -1 bzw. $+1$ liegen, liefert die Maschinenauswertung der rechten Seite in (1.17) brauchbare Ergebnisse. Gilt jedoch $\inf(x) = -1$ oder $\sup(x) = +1$, so erhält man mit dem gleichen Term erhebliche Intervallüberschätzungen, so daß die mit `Stagg_prec` geforderte Genauigkeit meist nicht erreicht werden kann. Mit $\sup(x) = +1$ erhält man daher mit einem schmalen Intervall x für das Intervall $\eta := 1 - x$ die Aussage:

$$0 = \inf(\eta) \leq \sup(\eta) \ll 1, \quad \text{und wegen der Identität}$$

$$\left\{ t \,|\, t = \sqrt{1-\zeta^2},\ \zeta \in x \right\} \equiv \left\{ t \,|\, t = \sqrt{\zeta}\cdot\sqrt{2-\zeta},\ \zeta \in \eta = 1 - x \right\}$$

folgt wieder mit der Inklusionsmonotonie

$$\sqrt{1-x^2} \subseteq \sqrt{\eta}\cdot\sqrt{2-\eta}, \quad \eta := 1 - x; \tag{1.18}$$

Im Fall $\inf(x) = -1$ erhält man mit dem Intervall $\eta := 1 + x$ für $\sqrt{1-x^2}$ die gleiche in (1.18) angegebene Einschließung. Wir werden daher $\sqrt{1-x^2}$ durch Auswertung der rechten Seite von (1.17) oder (1.18) einschließen. Bei Anwendung von (1.18) können sich wegen Intervallüberschätzungen negative Werte ergeben. In diesem Fall wird die Untergrenze des Ergebnisintervalls mit Hilfe der Anweisung `eps := Staggintval(0,xi.sup)` auf Null gesetzt. Im Falle $\sup(x) \approx 1$ aber $\sup(x) < 1$ wird ebenfalls (1.18) angewandt, weil dann $(1-x)\cdot(1+x)$ zu dicht bei 0 liegt und im `real`–Format nicht genau genug ausgewertet werden kann. Entsprechendes gilt im Fall $\inf(x) \approx -1$.

1.5.2 Der Programm-Quelltext

```
function sqrt1mx2(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := sqrt1mx2(x) liefert Einschliessung von  sqrt(1-x^2) }
const c = -130;
var eps,d1,d2 : Stagg[-1..Stagg_prec];
   anders     : boolean;
   xi         : interval;
begin
   eps := 1 - x;  d1 := eps;
   anders := d1[1] + d1[-1] = 0;
   if not anders then
   begin
      eps := 1 + x;  d2 := eps;
      anders := d2[1] + d2[-1] = 0;
   end;
   if anders then
   begin
      eps := sqrt(eps) * sqrt((2-eps));
      if SIGN(eps[1]+eps[-1]) < 0 then
         begin
            xi := ival(eps);
            eps := Staggintval(0,xi.sup);
         end;
      sqrt1mx2 := eps;
   end       else
     if (expo(d1[1])<c) or (expo(d2[1])<c) then
        sqrt1mx2 := sqrt(d1) * sqrt(d2)    else
        sqrt1mx2 := sqrt( d1*d2 );
end; { sqrt1mx2 }
```

1.5.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
```

```
      repeat
         writeln('x: Stagg   = ?');   read(x);
         y := sqrt1mx2(x);
         writeln('sqrt1mx2(x)  enthalten in: ');
         writeln(y);  writeln;
      until false;
   end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);   main;
end.
```

erhält man die folgenden Ergebnisse:

```
Stagg_prec = ?
10      <enter>
x: Stagg   = ?
10 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 -1e-267    <enter>
sqrt1mx2(x)  enthalten in:
 4.472135954999E-134
 5.793928183473E-147
 3.746255247088E-160
 1.236719223051E-173
 4.485417944908E-186
 2.104185127560E-199
 9.798828828816E-212
 7.575645499390E-225
 1.635230154756E-238
 7.008506535445E-251
[  3.893900000000E-263,  3.894200000000E-263 ]

Stagg_prec = ?
2      <enter>
x: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
0.6 0 0      <enter>
sqrt1mx2(x)  enthalten in:
 8.000000000000E-001
-0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

```
Stagg_prec = ?
21     <enter>
x: Stagg   = ?
21 Stagg components and 1 interval are needed in this sequence
0.7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1e-267     <enter>
sqrt1mx2(x)  enthalten in:
 7.141428428542E-001
 8.499979993998E-014
 1.136726527876E-027
 6.171159902733E-040
 8.332084308827E-053
 6.582040644002E-066
 1.886258988213E-079
 5.328204182344E-092
 8.963226050852E-105
 1.529581546609E-118
 6.322465946787E-131
 8.396869162444E-144
 7.035789313071E-157
 2.278304054854E-170
 4.975028922336E-183
 9.959091960141E-196
 6.615132603750E-209
 2.291451060955E-222
 4.865513039375E-235
 7.406991813618E-248
 2.831087000000E-261
[ -0.000000000000E+000,  1.000000000000E-267 ]

Stagg_prec = ?
3
x: Stagg   = ?
3 Stagg components and 1 interval are needed in this sequence
1 0 0 [-1e-267,0]
sqrt1mx2(x)  enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  4.472135955001E-134 ]
```

1.6 $\sqrt{x^2-1}$

1.6.1 Der Algorithmus

Mit dem staggered Intervallargument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ gilt zunächst die Definition:

$$\sqrt{x^2-1} := \left\{ t \,|\, t = \sqrt{\zeta^2-1};\; \zeta \in x \subseteq (-\infty, -1] \wedge \zeta \in x \subseteq [+1, +\infty) \right\}$$

Da $\sqrt{\zeta^2-1}$ eine gerade Funktion ist, kann man sich bei negativen ζ-Werten nach Multiplikation von x mit (-1) beschränken auf:

$$\sqrt{x^2-1} := \left\{ t \,|\, t = \sqrt{\zeta^2-1};\; \zeta \in x \subseteq [+1, +\infty) \right\}$$

Liegt nun $\sup(x)$ sehr dicht bei $+1$, so gilt mit $\eta := x - 1$

$$\sqrt{x^2-1} \subseteq \sqrt{\eta} \cdot \sqrt{2+\eta}, \quad \eta = x - 1 \tag{1.19}$$

Im Falle $\inf(x) \gg 1$ benutzen wir die Approximation

$$\sqrt{x^2-1} \approx x, \quad \text{falls } \inf(x) \gg 1, \tag{1.20}$$

wobei der Approximationsfehler entsprechend zu berücksichtigen ist. In den restlichen Fällen gilt:

$$\sqrt{x^2-1} \subseteq \sqrt{(x-1)\cdot(x+1)} \tag{1.21}$$

Da in (1.19) und (1.21) die rechten Seiten auf der Maschine intervallmäßig ausgewertet werden, ist damit in diesen Fällen die jeweilige Einschließung von $\sqrt{x^2-1}$ gesichert. Sollten sich dabei in (1.19) negative Funktionswerte ergeben, so wird die Unterschranke des Ergebnisintervalls auf Null gesetzt.

Wenn in (1.21) $\zeta \in x$ zu groß ist, liefert die Auswertung von $(x-1)\cdot(x+1)$ einen Overflow, der sich jedoch vermeiden läßt, wenn man die Näherung (1.20) benutzt. Da x jedoch nur eine Approximation von $\sqrt{x^2-1}$ ist, wird zunächst eine Einschließung für $\sqrt{x^2-1}$ angegeben, die für alle Argumentintervalle $x = [\inf(x), \sup(x)]$, mit $\inf(x) \geq 1$ gültig ist:

$$\sqrt{x^2-1} \subset \left[\frac{-1}{\inf(x)}, 0 \right] + x, \quad x = [\inf(x), \sup(x)] \tag{1.22}$$

Zum Beweis: Mit $\zeta \in x$ gilt zunächst $\sqrt{\zeta^2-1} \leq \sqrt{\sup^2(x)-1} < \sup(x)$. Andererseits bestätigt man für $\zeta \geq 1$ sehr einfach die Ungleichung:

$$\zeta - \frac{1}{\zeta} \leq \sqrt{\zeta^2-1}, \quad \zeta \geq 1 \tag{1.23}$$

und da $\sqrt{\zeta^2 - 1}$ in ζ monoton ist, erhält man mit (1.23) für $\inf(x) \geq 1$:

$$\inf(x) - \frac{1}{\inf(x)} \leq \sqrt{\inf^2(x) - 1} \leq \sqrt{\zeta^2 - 1} < \sup(x), \quad \text{d.h.}$$

$$\sqrt{x^2 - 1} \subset \left[\inf(x) - \frac{1}{\inf(x)}, \sup(x)\right] = \left[\frac{-1}{\inf(x)}, 0\right] + x, \quad \text{q.e.d.} \tag{1.24}$$

Für $x = [2, 2]$ erhält man damit z.B. die Einschließung $\sqrt{[3, 3]} \subset [3/2, 2]$, die natürlich viel zu grob ist. Die für alle Intervalle $x = [\inf(x), \sup(x)]$, mit $\inf(x) \geq 1$ gültige Einschließung (1.24) ist nur interessant für $\inf(x) \ggg 1$, denn der Betrag F_0 des absoluten Fehlers läßt sich für $\zeta_0 := \inf(x)$ wie folgt abschätzen:

$$F_0 := \sqrt{\zeta_0^2 - 1} - \left(\zeta_0 - \frac{1}{\zeta_0}\right) = \sqrt{\zeta_0^2 - 1} - \zeta_0 + \frac{1}{\zeta_0} \leq \frac{1}{\zeta_0},$$

d.h. F_0 wird für $\zeta_0 \to \infty$ beliebig klein. Da man bzgl. $\zeta_1 := \sup(x)$ für den Betrag F_1 des absoluten Fehlers nach (1.23) die gleich Abschätzung

$$F_1 := \zeta_1 - \sqrt{\zeta_1^2 - 1} \leq \frac{1}{\zeta_1},$$

erhält, wird die Einschließung (1.24) umso genauer, je größer $\inf(x)$ gewählt wird. Um für alle vorgegebenen Präzisionsparameter $S =$ `Stagg_prec` möglichst enge Einschließungen von $\sqrt{x^2 - 1}$ zu erhalten, kommt (1.24) nur dann zur Anwendung, wenn bei der Einschließung nach (1.21) die Auswertung von $(x - 1) \cdot (x + 1)$ zum Overflow führt. Im Programm wird die entsprechende Abfrage realisiert durch:

```
if expo(t[1] + t[0]) > 127 then
begin { Overflow in (t-1)*(t+1) ist moeglich, }
      { daher Anwendung der Einschliessung:   }
      { sqrt(x^2-1) enthalten in x + [-1/inf(x),0] }
   zx := #<( t[-1] + for j:=1 to ub(t) sum(t[j]) );
   zx := 1 /> zx;
   t := t + Staggintval(-zx,0);
end
```

Es gilt die Beziehung: $\quad 1 < \texttt{t}[1] + \texttt{t}[0] \leq \sup(x).$

1.6.2 Der Programm-Quelltext

```
function sqrtx2m1(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := sqrtx2m1(x) liefert Einschliessung von  sqrt(x^2-1) }
const c = -125;
var t,eta : Stagg[-1..Stagg_prec];
   zx     : real;
   xi     : interval;
   j      : integer;
begin
   t := x;  normal(t);
   if SIGN(t[1] + t[-1]) < 0 then t := -t;
   if expo(t[1] + t[0]) > 127 then
   begin
      zx := #<( t[-1] + for j:=1 to ub(t) sum(t[j]) );
      zx := 1 /> zx;
      t := t + Staggintval(-zx,0);
   end  else
   begin
      eta := t - 1;
      zx := eta[1] + eta[0];
      if expo(zx) < c then
      begin
         t := sqrt(eta)*sqrt(eta+2);
         if SIGN(t[1]+t[-1]) < 0 then
         begin { Hier wird verhindert, dass Elemente aus }
            xi := ival(t);          { t negativ werden. }
            t := Staggintval(0,xi.sup);
         end;
      end             else t := sqrt( (t-1)*(t+1) );
   end;
   sqrtx2m1 := t;
end; { sqrtx2m1 }
```

Beachten Sie bitte, das mit dem Funktionsaufruf `Staggintval(-zx,0)` eine garantiert Einschließung von $[-1/\zeta_0, 0]$ berechnet wird, um die Einschließung von $\sqrt{x^2-1}$ in (1.24) garantieren zu können!

1.6.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
{ Einschliessung von sqrt(x^2-1) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := sqrtx2m1(x);
       writeln('sqrtx2m1(x)  enthalten in: ');
       write(y);
       writeln;
    until false;
end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man zur Einschließung von $\sqrt{x^2-1}$ die folgenden Ergebnisse:

```
Stagg_prec = ?
6      <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 1e-267  <enter>
sqrtx2m1(x)  enthalten in:
 4.472135954999E-134
 5.793928183473E-147
 3.746255247088E-160
 1.236719223051E-173
 4.485417944908E-186
 2.104185127560E-199
[  9.798828828808E-212,  9.798828828820E-212 ]
```

```
Stagg_prec = ?
1      <enter>
x: Stagg   = ?
1 Stagg components and 1 interval are needed in this sequence
1 [0,1e-267]       <enter>
sqrtx2m1(x)  enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  4.472135955001E-134 ]

Stagg_prec = ?
22     <enter>
x: Stagg   = ?
22 Stagg components and 1 interval are needed in this sequence
9.999e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
sqrtx2m1(x)  enthalten in:
 9.998999999999E+255
 9.999999999999E+242
 9.999999999999E+229
 9.999999999999E+216
 9.999999999999E+203
 9.999999999999E+190
 9.999999999999E+177
 9.999999999999E+164
 9.999999999999E+151
 9.999999999999E+138
 9.999999999999E+125
 9.999999999999E+112
 9.999999999999E+099
 9.999999999999E+086
 9.999999999999E+073
 9.999999999999E+060
 9.999999999999E+047
 9.999999999999E+034
 9.999999999999E+021
 9.999999999999E+008
 9.999999999999E-005
 9.999999999999E-018
[  9.999999999999E-031,  1.000000000000E-030 ]
```

```
Stagg_prec = ?
21     <enter>
x: Stagg   = ?
21 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0          <enter>
sqrtx2m1(x)  enthalten in:
 1.732050807568E+000
 8.772935274463E-013
 4.150587236694E-026
 2.805253810380E-039
 6.280558069794E-052
 5.193301690880E-065
 3.708114618675E-080
 7.248575675626E-093
 1.414154067030E-106
 2.996994509499E-119
 8.952478811655E-132
 5.120943736485E-145
 2.809323190230E-158
 5.582067974820E-171
 1.010846749232E-184
 6.501531234326E-197
 6.903322886650E-210
 6.722546689218E-223
 3.797122704713E-236
 1.660367861588E-249
 1.904900000000E-263
[ -0.000000000000E+000,  1.000000000000E-267 ]

Stagg_prec = ?
4
x: Stagg   = ?
4 Stagg components and 1 interval are needed in this sequence
2 0 0 0 [1e-50,2e-50]
sqrtx2m1(x)  enthalten in:
 1.732050807568E+000
 8.772935274463E-013
 4.150587236694E-026
 0.000000000000E+000
[  2.805253810392E-039,  2.805253810404E-039 ]
```

1.7 $\sqrt{x^2+1}$

1.7.1 Der Algorithmus

Mit dem staggered Intervallargument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ gilt zunächst die Definition:

$$\sqrt{x^2+1} := \left\{ t \,|\, t = \sqrt{\zeta^2+1};\ \zeta \in x \subseteq (-\infty, +\infty) \right\}$$

Da $\sqrt{\zeta^2+1}$ eine gerade Funktion ist, kann man sich im Algorithmus beim Auftreten negativer ζ-Werte beschränken auf das Intervallargument $|x|$.

$$\sqrt{x^2+1} := \left\{ t \,|\, t = \sqrt{\zeta^2+1};\ \zeta \in x \subseteq [0, +\infty) \right\}$$

Im Falle $\inf(x) \gg 1$ benutzen wir die Approximation

$$\sqrt{x^2+1} \approx x, \quad \text{falls } \inf(x) \gg 1, \tag{1.25}$$

wobei der Approximationsfehler entsprechend zu berücksichtigen ist; Sonst gilt:

$$\sqrt{x^2+1} \subseteq \sqrt{x \cdot x + 1} \tag{1.26}$$

Da in (1.26) die rechte Seite auf der Maschine intervallmäßig ausgewertet wird, ist damit die Einschließung von $\sqrt{x^2+1}$ gesichert.

Wenn in (1.26) $\zeta \in x$ zu groß ist, liefert die Auswertung von $x \cdot x + 1$ einen Overflow, der sich jedoch vermeiden läßt, wenn man die Näherung (1.25) benutzt. Da x jedoch nur eine Approximation von $\sqrt{x^2+1}$ ist, wird zunächst eine Einschließung für $\sqrt{x^2+1}$ angegeben, die für alle Argumentintervalle $x = [\inf(x), \sup(x)]$, mit $\inf(x) > 0$ gültig ist:

$$\sqrt{x^2+1} \subset \left[0, \frac{1}{2 \cdot \sup(x)}\right] + x, \quad x = [\inf(x), \sup(x)] \tag{1.27}$$

Zum **Beweis:** Mit $\zeta \in x$ gilt zunächst: $\inf(x) < \sqrt{1 + \inf^2(x)} \leq \sqrt{1+\zeta^2}$. Andererseits bestätigt man für $\zeta > 0$ sehr einfach die Ungleichung:

$$0 < \zeta < \sqrt{\zeta^2+1} < \zeta + \frac{1}{2\zeta}, \quad \zeta > 0 \tag{1.28}$$

und da $\sqrt{\zeta^2+1}$ in ζ monoton ist, erhält man mit (1.28) für $\inf(x) > 0$:

$$\sqrt{1+\zeta^2} \leq \sqrt{1 + \sup^2(x)} < \sup(x) + \frac{1}{2 \cdot \sup(x)}, \qquad \text{und damit:}$$

$$\inf(x) < \sqrt{\zeta^2+1} < \sup(x) + \frac{1}{2\cdot\sup(x)}, \quad \zeta \in x = [\inf(x), \sup(x)],$$

und daraus ergibt sich die Behauptung (1.27) unmittelbar. Für $x = [2,2]$ erhält man damit z.B. die Einschließung $\sqrt{[5,5]} \subset [2, 2.25]$, die natürlich viel zu grob ist. Die für alle Intervalle $x = [\inf(x), \sup(x)]$, mit $\inf(x) > 0$ gültige Einschließung (1.27) ist nur interessant für $\inf(x) \ggg 1$, denn der Betrag F_0 des absoluten Fehlers läßt sich für $\zeta_0 := \inf(x)$ wie folgt abschätzen:

$$F_0 := \sqrt{1+\zeta_0^2} - \zeta_0 < \frac{1}{2\zeta_0}$$

d.h. F_0 wird für $\zeta_0 \to \infty$ beliebig klein. Da man bzgl. $\zeta_1 := \sup(x)$ für den Betrag F_1 des absoluten Fehlers nach (1.28) die gleich Abschätzung

$$F_1 := \zeta_1 + \frac{1}{2\zeta_1} - \sqrt{\zeta_1^2+1} < \frac{1}{2\zeta_1},$$

erhält, wird die Einschließung (1.27) umso genauer, je größer $\inf(x)$ gewählt wird. Um für alle vorgegebenen Präzisionsparameter $S =$ `Stagg_prec` möglichst enge Einschließungen von $\sqrt{x^2+1}$ zu erhalten, kommt (1.27) nur dann zur Anwendung, wenn bei der Einschließung nach (1.26) die Auswertung von $x \cdot x + 1$ zum Overflow führt. Im Programm wird die entsprechende Abfrage realisiert durch:

```
if expo(t[1]+t[0]) > 127 then
   begin  { Overflow in t*t+1 ist moeglich, }
      { daher Anwendung der Einschliessung: }
      { sqrt(x^2+1) enthalten in x + [0,1/(2*inf(x))] }
      zx := #<( for j:=0 to ub(t) sum(t[j]) );
      zx := 0.5 /> zx;
      t := t + Staggintval(0,zx);
   end                    else t := sqrt(t*t+1);
```

Es gilt die Beziehung: $\quad 0 < \mathtt{t}[1] + \mathtt{t}[0] \le \sup(x).$

1.7.2 Der Programm-Quelltext

```
function sqrtx2p1(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := sqrtx2p1(x) liefert Einschliessung von sqrt(x^2+1) }
var t : Stagg[-1..Stagg_prec];
   zx : real;
   j  : integer;
begin
   t := x;  normal(t);
   if SIGN(t[1]+t[-1]) < 0 then t := abs(t);
   if expo(t[1]+t[0]) > 127 then
   begin
      zx := #<( for j:=0 to ub(t) sum(t[j]) );
      zx := 0.5 /> zx;
      t := t + Staggintval(0,zx);
   end                      else t := sqrt(t*t+1);
   sqrtx2p1 := t;
end; { sqrtx2p1 }
```

1.7.3 Numerische Beispiele

Mit dem **PASCAL-XSC** Programm

```
program tests;
{ Einschliessung von sqrt(x^2+1) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := sqrtx2p1(x);
       writeln('sqrtx2p1(x)  enthalten in: ');
       write(y);  writeln;
    until false;
end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);          main;
end.
```

erhält man zur Einschließung von $\sqrt{x^2+1}$ die folgenden Ergebnisse:

```
Stagg_prec = ?
2      <enter>
x: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
9.999999999999e255 0 1e-267       <enter>
sqrtx2p1(x)  enthalten in:
 9.999999999999E+255
 0.000000000000E+000
[  1.000000000000E-267,  5.000000000200E-257 ]

Stagg_prec = ?
21     <enter>
x: Stagg   = ?
21 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0           <enter>
sqrtx2p1(x)  enthalten in:
 2.236067977499E+000
 7.896964091736E-013
 6.873127623544E-026
 6.183596115257E-040
 2.427089724541E-053
 5.209256378048E-067
 9.941441440837E-080
 8.782274969508E-093
 1.761507737835E-106
 4.253267724447E-120
 7.386358636012E-134
 1.533452708866E-147
 7.781731918791E-160
 6.581127664532E-173
 2.639856580535E-186
 7.613504175337E-199
 8.500342339241E-212
 4.064442086432E-225
 5.390972525926E-238
 2.722887629951E-251
 7.402000000000E-264
[ -0.000000000000E+000,  1.000000000000E-267 ]
```

1.8 Die n-te Wurzel

1.8.1 Der Algorithmus

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist $\sqrt[n]{x}$ definiert durch:

$$\sqrt[n]{x} := \left\{ t \,|\, t = \sqrt[n]{\zeta},\ \zeta \in x \right\}, \qquad n = 2, 3, \ldots$$

Bei geradzahligem n gilt die einschkränkende Bedingung $\mathrm{Inf}(x) \geq 0$. Gesucht ist eine staggered Variable z, die $\sqrt[n]{x}$ garantiert einschließt:

$$\sqrt[n]{x} \subseteq z = z_1 + z_2 + \ldots + z_S + [z_{-1}, z_0]$$

Der Präzisionsparameter $S =$ `Stagg_prec` wird vom Anwender vorgegeben und legt durch $13 \cdot (S + 1)$ die Anzahl der benutzten Dezimalstellen in der staggered Intervall–Arithmetik fest. Im nachfolgenden Algorithmus können S und $\mathrm{ub}(x)$ verschiedene Werte annehmen.

Die Einschließung von $\sqrt[n]{x}$ kann zurückgeführt werden auf die Einschließung der Nullstelle folgender Funktion:

$$f(y) := y^n - x, \qquad n = 2, 3, \ldots$$

Nach dem Intervall–Newton-Verfahren erhält man eine Einschließung dieser Nullstelle durch folgenden Algorithmus:

$$[y]_0 := \Diamond(\sqrt[n]{x})$$

$$[y]_{k+1} := \mathrm{mid}([y]_k) - \frac{\big(\mathrm{mid}([y]_k)\big)^n - x}{n \cdot [y]_k^{n-1}}, \quad k = 0, 1, \ldots \tag{1.29}$$

Das staggered Startintervall $[y]_0$ erhält man zunächst durch Einschließung von x mit einem Intervall `v` vom Typ `interval` durch `v := ival(x)`, und `v := power(v,intval(1)/n)` liefert dann im einfachen `real`–Format eine Einschließung von $\sqrt[n]{x}$. Im Programm wird dann das Startintervall $[y]_0$ durch `y := v` realisiert, das $\sqrt[n]{x}$ mit nur 12 richtigen Dezimalziffern zunächst nur grob einschließt. Eine Verbesserung dieser Einschließung erhält man in den durch (1.29) definierten Newtonschritten, wobei die Präzision mit Hilfe der `uu[k]` solange erhöht wird, bis die durch `Stagg_prec` vorgegebene Präzision erreicht wird. Wie bei der Quadratwurzel wird auch jetzt das Argument x geeignet skaliert, um mögliche Fehlermeldungen bei der staggered Division zu vermeiden.

1.8.2 Der Programm-Quelltext

```
function sqrtn_ge0(var x : Stagg; n : integer)
                                  : Stagg[-1..Stagg_prec];
{ Einschliessung der n-ten Wurzel fuer 0 <= x <= 9.9E+255 }
{ ub(x) <> Staggg_prec ist zulaessig.                     }
{ Blomquist, 29.01.2001;                                  }
var v            : interval;
   y,m           : Stagg;
   ergb          : Stagg[-1..Stagg_prec];
   xx            : Stagg[-1..ub(x)];
   k,p,j,S,a,b : integer;
   ready         : boolean;
   rel           : real;
   uu            : array[1..7] of integer;
begin
   k := expo(0);
   for j := -1 to ub(x) do
   begin
      p := expo(x[j]);
      if p > k then k := p;
   end;       { k:  maximaler Zehnerexponent der x[j] }
   { Berechnung der Skalierungs-Exponenten   a,b:       }
   b := (254-k) div (-n);      { b: zur Rueckskalierung }
   a := -n*b;                  { a: zur Startskalierung }
   xx := x; { Laesst Argument x unveraendert!           }
   { Startskalierung von xx mit dem Zehner-Exponenten a }
   for j := -1 to ub(x) do xx[j] := xx[j] M_10hoch a;
   v := ival(xx);  { v: Einschliessung des skalierten x }
   if v=0 then ergb := 0 else
   begin
      k := expo(v.sup);  { k: Exponent des skalierten x }
      S := k div n;
      v := v * comp(0.1,-n*S+1);     { v: Einschliessg. }
      v := power(v,Intval(1)/n);     { der n-ten Wurzel }
      v := comp(0.1,S+1) * v;
      { Die uu[k] liefern die aktuelle Praezision im k-ten }
      { Newton-Schritt der nachfolgenden repeat-Schleife.  }
      uu[1] := 1;  uu[2] := 1;  uu[3] := 2;
```

```
      uu[4] := 4;  uu[5] := 8;  uu[6] := 16;  uu[7] := 32;
      S := Stagg_prec;  rel := 0;
      if diam(v) > 0 then rel := (v.sup->v.inf)/v.sup;
      if (diam(v)>0) and (rel<2e-12) then
      begin   { Intervall-Newton-Verfahren mit }
         k := 0;        { variabler Praezision: }
         allocate(y,-1..1); allocate(m,-1..1);
         y := v;
         repeat
            k := k+1;
            ready := uu[k] >= S;
            if ready then uu[k] := S;
            set_Stagg_prec(uu[k]);   { uu[k]: Neue Praezision }
            allocate(m,-1..Stagg_prec);  m := 0;
            for j := -1 to ub(y) do m[j] := y[j];
            allocate(y,-1..Stagg_prec);
            y := m;   { altes y jetzt in neuer Praezision vor }
            if k=2 then m := SUP(y) else m := mid(y);
            y := m - (power(m,n)-xx)/(n*power(y,n-1));
            if y[-1]=y[0] then
            begin
               allocate(m,-1..S);   m := 0;
               for j := -1 to ub(y) do m[j] := y[j];
               allocate(y,-1..S);
               y := m; { y jetzt wieder in Ausgangspraezision }
               ready := true; { PUNKTeinschliessung gefunden: }
            end;              { --->   ready := TRUE         }
         until ready;
         ergb := y;  free(y);  free(m);
      end else ergb := v;
      { Jetzt folgt die Rueckskalierung mit dem Exponenten b: }
      set_stagg_prec(S);
      ergb := comp(0.1,b+1) * ergb;
   end;
   sqrtn_ge0 := ergb;
end; { sqrtn_ge0 }

function sqrtn(var x : Stagg; n: integer)
                                       : Stagg[-1..Stagg_prec];
var lo,up,Null : Stagg[-1..Stagg_prec];
```

```
   dot         : dotprecision;
   p           : integer;
begin
   if n < 2 then s_fu1_math_error(5);
   if Stagg_prec > 30 then s_fu1_math_error(7);
   if n mod 2 = 0 then
   begin
      dot := #(x[-1] + for p:=1 to ub(x) sum(x[p]));
      if SIGN(dot) < 0 then s_fu1_math_error(6);
      sqrtn := sqrtn_ge0(x,n);
   end            else
   begin
      p := 0;
      if x >= 0 then p := 1 else
         if x <= 0 then p := -1;
      case p of
        +1 : sqrtn := sqrtn_ge0(x,n);
        -1 : sqrtn := -sqrtn_ge0(-x,n);
        0  : begin { Null liegt echt im Innern von x }
                Null := 0;
                up := hull(Null,SUP(x));
                up := sqrtn_ge0(up,n);
                lo := -hull(INF(x),Null);
                lo := -sqrtn_ge0(lo,n);
                sqrtn := hull(INF(lo),SUP(up));
             end;
      end;  { case }
   end;
end; { sqrtn }
```

Wenn $n < 2$ oder `Stagg_prec` > 30 gewählt werden, so erfolgt eine entsprechende Fehlermeldung. Ist n gerade und enthält das Argument x negative Elemente, so erfolgt ebenfalls eine Fehlermeldung mit Programmabbruch. Bei ungeradem n werden in obiger `case`–Anweisung drei Fälle unterschieden, je nachdem, ob $\mathrm{Inf}(x) \geq 0$ oder $\mathrm{Sup}(x) \leq 0$ oder ob die Null im Innern des Arguments x liegt. Im letzten Fall muß die n-te Wurzel recht aufwendig eingeschlossen werden; in der numerischen Praxix spielt dieser Fall jedoch keine Rolle!

1.8.3 Numerische Ergebnisse

Mit dem Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 n       : integer;
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       writeln('n: integer = ?');   read(n);
       y := sqrtn(x,n);
       writeln('sqrtn(x,n)  enthalten in: ');
       writeln(y);
    until false;
 end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man die folgenden Ergebnisse:

```
Stagg_prec = ?
5      <enter>
x: Stagg   = ?
 5 Stagg components and 1 interval are needed in this sequence
1e-251 0 0 0 0 0   <enter>
n: integer = ?
100000             <enter>
sqrtn(x,n)  enthalten in:
 9.942371805322E-001
 7.948186189587E-014
-0.000000000000E+000
-0.000000000000E+000
-0.000000000000E+000
[ -9.942400000000E-022,  9.942400000000E-022 ]
```

```
Stagg_prec = ?
30     <enter>
x: Stagg   = ?
30 Stagg components and 1 interval are needed in this sequence
1e254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1e-267   <enter>
n: integer = ?
5             <enter>
sqrtn(x,n)  enthalten in:
 6.309573444801E+050
 9.324943436013E+037
 6.622343864672E+024
 9.452571882287E+011
 2.452772952883E-002
 3.494943297686E-015
 8.075729140677E-028
 4.365226487327E-041
 3.467309883646E-054
 5.906764239646E-067
 4.880448569818E-080
 5.465559246598E-093
 5.036241859191E-106
 3.395239151931E-119
 8.659563206889E-132
 2.720998767238E-145
 3.311470968899E-158
 7.016679402002E-171
 6.058481150123E-184
 2.703584524931E-197
 1.315549845097E-210
 9.927430411981E-223
 3.120640770920E-237
 7.179126950029E-250
 7.253500000000E-263
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  1.000000000000E-267 ]
```

Das vorhergehende Beispiel zeigt, daß man mit wachsenden `Stagg_prec`–Werten nicht beliebige Genauigkeiten erreichen kann, da im `real`–Format die kleinste positive Zahl durch 10^{-267} gegeben ist. Größere Genauigkeiten lassen sich daher nur erreichen, wenn man die auf dem Datentyp `Longreal` basierende staggered Arithmetik benutzt. Im `Longreal`–Format ist nämlich die kleinste positive Zahl gegeben durch 10^{-531}. Die Standardfunktionen in diesem staggered Format werden vom Modul `ls_fu1` exportiert, und die eigentliche staggered Arithmetik wird durch das Modul `lstagg` realisiert.

```
Stagg_prec = ?
20     <enter>
x: Stagg   = ?
20 Stagg components and 1 interval are needed in this sequence
1.2e32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
n: integer = ?
10000  <enter>
sqrtn(x,n)  enthalten in:
 1.007413851969E+000
 9.966599606719E-013
 1.523013401103E-026
 7.562961547752E-039
 3.292883236981E-053
 8.225225273342E-066
 2.790617259207E-079
 4.704864568090E-093
 4.069997871174E-106
 5.364255255967E-119
 1.421830851998E-132
 9.879081348237E-145
 2.502269800818E-158
 8.283083478478E-171
 5.335457429115E-184
 7.770589543158E-197
 7.368976841692E-210
 9.133861604756E-223
 2.244193107566E-236
 2.425051638825E-249
[  9.240740000000E-262,  9.240780000000E-262 ]
```

1.9 Die Logarithmusfunktionen

Zum vorgegebenen staggered Intervallargument

$$x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$$

berechnet man im Dezimalsystem von **PASCAL-XSC** zunächst eine Einschließung der Logrithmus–Funktion $\log_{10}(x)$ zur Basis 10, wobei $\log_{10}(x)$ definiert ist durch:

$$\log_{10}(x) := \{y \mid y = \log_{10}(\zeta),\ \zeta \in x\} \tag{1.30}$$

Das Intervall $\log_{10}(x)$ ist daher der Wertebereich der reellen Logarithmusfunktion $\log_{10}(\zeta)$, $\zeta > 0$ über dem Intervall $x \ni \zeta$. Schreibt man das reelle Argument als Produkt

$$\zeta = m_\zeta \cdot 10^{ex}, \qquad 0.1 \le m_\zeta < 1, \quad ex = 0. \pm 1, \pm 2, \ldots\ , \tag{1.31}$$

so gilt im Dezimalsystem

$$\log_{10}(\zeta) = ex + \log_{10}(m_\zeta)$$

Damit hat man ein beliebiges Argument $\zeta > 0$ schon auf das schmale Intervall $[0.1, 1)$ reduziert, was den nachfolgenden Algorithmus zur Einschließung von $\log_{10}(m_\zeta)$ deutlich vereinfacht.

Der natürliche Logarithmus zur Basis e ergibt sich dann zu:

$$\ln(x) \equiv \log_{10}(x) \cdot \ln(10) \tag{1.32}$$

Die staggered Konstante `ln10`, die $\ln(10)$ maximalgenau einschließt, wird im Modul `s_fu1` bereitgestellt. Der laufzeitmäßige Nachteil der rechts in (1.32) notwendigen Multiplikation wird durch die mit (1.31) realisierten Argumentreduktion mehr als wettgemacht.

Beachten Sie bitte, daß im nachfolgenden Abschnitt zwischen den reellen Größen ζ, m_ζ und den staggered Intervallen x, m, t streng zu unterscheiden ist! Der mit der Intervallrechnung noch nicht so vertraute Leser sollte den folgenden Abschnitt sorgfältig studieren. Ich habe versucht, die kritischen Punkte ausführlich darzulegen und die entsprechenden Schritte im Algorithmus entsprechend zu begründen. Mit Hilfe des leicht lesbaren **PASCAL-XSC** Quelltextes ab Seite 55 sollt der Leser den beschriebenen Algorithmus nachvollziehen können. Die Grundlagen der Intervallrechnung, wie z.B. die oft benutzte Inklusionsmonotonie, findet der Leser in [2].

1.9.1 $\log_{10}(x)$

Zur Vereinfachung gilt im folgenden Abschnitt die Schreibweise $\log_{10} \equiv \log$.

1.9.1.1 Der Algorithmus

Falls der relative Durchmesser des staggered Intervallarguments

$$x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$$

größer ist als 10^{-14}, erfolgt die Einschließung von $\log(x)$ im einfachen `real`–Format mit Hilfe der Anweisung:

```
t := Staggintval( log10(ival(x)) );
```

`ival(x)` liefert zunächst ein Intervall vom Typ `interval`, welches x optimal einschließt, und `log10(ival(x))` liefert eine garantierte Einschließung von $\log(x)$ im einfachen `real`–Format. Die Funktion `Staggintval` transformiert dann diese Einschließung ins benutzte staggered Format, dessen Präzision durch den Parameter `Stagg_prec` vom Anwender festgelegt wird. Es gilt dann: $\log(x) \subseteq \mathtt{t}$, wobei $\log(x)$ nach (1.30) definiert ist durch:

$$\log(x) := \{y \,|\, y = \log(\zeta),\ \zeta \in x\}$$

Falls der relative Durchmesser von x kleiner ist als 10^{-14}, wird für dieses schmale Intervall im ersten Schritt eine Produktzerlegung realisisert. Mit Hilfe der Anweisung ex `:= expo(x[1]+x[0])` bestimmt man zunächst den Zehnerexponenten ex und mit `m := x M_10HOCH` $-ex$ die zugehörige Intervallmantisse `m`, so daß gilt:

$$x \subseteq \mathtt{m} \cdot 10^{ex}, \qquad \mathtt{m} \Subset [0.1, 1], \quad ex = 0, \pm 1, \pm 2, \ldots\ ; \tag{1.33}$$

Dabei bedeutet $\mathtt{m} \Subset [0.1, 1]$, daß Elemente aus `m` auch geringfügig kleiner als 0.1 oder auch geringfügig größer als 1 ausfallen können. Später werden wir erkennen, daß diese minimalen Grenzüberschreitungen auf die korrekte Einschließung von $\log(x)$ keinerlei Einfluß hat! Nach [2, Seite 31] gilt dann:

$$\log(x) \subseteq \log(\mathtt{m} \cdot 10^{ex}) \tag{1.34}$$

und wie bei reeller Rechnung gilt jetzt auch für Intervalle die Beziehung:

$$\log(\mathtt{m} \cdot 10^{ex}) = ex + \log(\mathtt{m}),$$

denn nach Definition gilt:

$$\begin{aligned}\log(\mathtt{m}\cdot 10^{ex}) &= \{y\,|\,y=\log(\zeta\cdot 10^{ex}),\ \zeta\in\mathtt{m}\}\\ &= \{y\,|\,y=ex+\log(\zeta),\ \zeta\in\mathtt{m}\}\\ &= ex+\{y\,|\,y=\log(\zeta),\ \zeta\in\mathtt{m}\}\\ &= ex+\log(\mathtt{m})\end{aligned}$$

Mit (1.34) folgt dann direkt:

$$\log(x)\subseteq ex+\log(\mathtt{m}),\qquad \mathtt{m}\Subset[0.1,1] \tag{1.35}$$

Zu berechnen bleibt also eine Einschließung von $\log(\mathtt{m})$, wobei $\mathtt{m}$ ein recht schmales staggered Intervall ist. Mit Hilfe der Potenzreihe der Logarithmusfunktion gilt nach Definition:

$$\log(\mathtt{m})=\left\{y\,|\,y=\frac{2r}{\ln(10)}\sum_{k=0}^{\infty}\frac{r^{2k}}{2k+1};\ r=\frac{\zeta-1}{\zeta+1},\ \zeta\in\mathtt{m}\right\} \tag{1.36}$$

und wegen $\mathtt{m}\Subset[0.1,1]$ kann r den Wert $-0.9/1.1$ annehmen, der für eine gute Konvergenz der Reihe betragsmäßig noch viel zu groß ist. Das staggered Argument $\mathtt{m}$ sollte daher möglichst dicht bei 1 liegen, damit die r-Werte betragsmäßig hinreichend klein werden. Dies erreicht man nun mit Hilfe der Transformation

$$\log(\mathtt{m})=p\cdot\log(\sqrt[p]{\mathtt{m}}),\quad p=128^k,\ k=1,2,3,\quad \mathtt{m}\Subset[0.1,1], \tag{1.37}$$

denn wählt man p hinreichend groß, so liegt das Intervall $\sqrt[p]{\mathtt{m}}$ hinreichend dicht bei 1. Daß die aus dem Reellen bekannte Beziehung (1.37) tatsächlich auch für Intervalle $\mathtt{m}$ gilt, zeigt folgende Überlegung:

$$\begin{aligned}p\cdot\log(\sqrt[p]{\mathtt{m}}) &= p\cdot\left\{y\,|\,y=\log(\sqrt[p]{\zeta}),\ \zeta\in\mathtt{m}\right\}\\ &= p\cdot\left\{y\,|\,y=\frac{1}{p}\log(\zeta),\ \zeta\in\mathtt{m}\right\}\\ &= \{y\,|\,y=\log(\zeta),\ \zeta\in\mathtt{m}\}=\log(\mathtt{m}),\quad \text{q.e.d.}\end{aligned}$$

Mit (1.35) folgt dann:

$$\log(x)\subseteq ex+p\cdot\log(\sqrt[p]{\mathtt{m}}),\quad p=128^k,\ k=1,2,3,\ \mathtt{m}\Subset[0.1,1] \tag{1.38}$$

Die Einschließung des Intervallarguments $\sqrt[p]{\mathtt{m}}\subseteq t$ erfolgt im Programm mit Hilfe der Anweisung `t := sqrt128(m)` , da die 128-te Wurzel mit einem

Minimum an Rechenoperationen realisiert werden kann. Auf den speziellen Algorithmus dieser Wurzeleinschließung wird hier nicht näher eingegangen, da man weitere Einzelheiten dazu ab Seite 41 nachlesen kann. Wenn es jedoch bei größerem Präzisionsparameter `Stagg_prec` notwendig wird, die 128^2-te Wurzel einzuschließen, so läßt sich dies mit der gleichen Funktion realisiert durch $\sqrt[128]{\sqrt[128]{\mathtt{m}}}$. Ganz entsprechende Überlegungen gelten, wenn die 128^3-te Wurzel einzuschließen ist. Für $k = 1, 2, 3$ gilt damit nach (1.38)

$$\log(x) \subseteq ex + p \cdot \log(\mathtt{t}), \quad \sqrt[p]{\mathtt{m}} \subseteq \mathtt{t},\ p = 128^k,\ \mathtt{m} \Subset [0.1, 1] \tag{1.39}$$

Approximation von $\log(\mathtt{t})$

Nach (1.36) gilt zusammen mit der Inklusionsmonotonie:

$$\begin{aligned} \log(\mathtt{t}) &:= \left\{ y \,|\, y = \frac{2r}{\ln(10)} \sum_{k=0}^{\infty} \frac{r^{2k}}{2k+1}; \quad r = \frac{\zeta - 1}{\zeta + 1},\ \zeta \in t \right\} \\ &\subseteq \frac{2}{\ln(10)} \cdot \frac{t-1}{t+1} \cdot \left\{ y \,|\, y = \sum_{k=0}^{\infty} \frac{r^{2k}}{2k+1}; \quad r = \frac{\zeta - 1}{\zeta + 1},\ \zeta \in t \right\} \qquad (1.40) \end{aligned}$$

und mit

$$f(\mathtt{t}) := \left\{ y \,|\, y = \sum_{k=0}^{\infty} \frac{r^{2k}}{2k+1}; \quad r = \frac{\zeta - 1}{\zeta + 1},\ \zeta \in t \right\} \quad \text{folgt} \tag{1.41}$$

$$\log(\mathtt{t}) \subseteq \frac{2}{\ln(10)} \cdot \frac{t-1}{t+1} \cdot f(\mathtt{t}) \tag{1.42}$$

Mit

$$f_N(\mathtt{t}) := \left\{ y \,|\, y = \sum_{k=0}^{N} \frac{r^{2k}}{2k+1}; \quad r = \frac{\zeta - 1}{\zeta + 1},\ \zeta \in t \right\} \tag{1.43}$$

definieren wir das Intervall $\varepsilon(\mathtt{t}, N)$ des Approximationsfehlers durch

$$f(\mathtt{t}) = f_N(\mathtt{t}) + \varepsilon(\mathtt{t}, N) \tag{1.44}$$

und erhalten:

$$\varepsilon(\mathtt{t}, N) = \left\{ y \,|\, y = \sum_{k=N+1}^{\infty} \frac{r^{2k}}{2k+1}; \quad r = \frac{\zeta - 1}{\zeta + 1},\ \zeta \in t \right\}$$

Damit ist $\varepsilon(\mathtt{t}, N)$ der Wertebereich der geraden Funktion

$$g(r) := \sum_{k=N+1}^{\infty} \frac{r^{2k}}{2k+1}, \quad r = \frac{\zeta - 1}{\zeta + 1}, \quad \text{über dem Intervall } \zeta \in \mathtt{t}$$

Bevor wir nun eine Einschließung dieses Wertebereichs angeben, soll zum Verständnis des Lesers ausdrücklich betont werden, daß auf der Maschine nicht das Intervall $f_N(\mathtt{t})$ nach (1.43) ausgewertet wird! Nach der Inklusionsmonotonie gilt für $f_N(\mathtt{t})$ die Einschließung:

$$f_N(\mathtt{t}) \subseteq P_N(\mathtt{v}) := \sum_{k=0}^{N} \frac{\mathtt{v}^{\mathtt{k}}}{2k+1}, \quad v = \left(\frac{t-1}{t+1}\right)^2 \tag{1.45}$$

und genau dieses $P_N(\mathtt{v})$ wird auf dem Rechner in staggered Arithmetik ausgewertet. Mit (1.42) und (1.44) erhalten wir daher

$$\log(\mathtt{t}) \subseteq \frac{2}{\ln(10)} \cdot \frac{t-1}{t+1} \cdot [P_N(\mathtt{v}) + \varepsilon(\mathtt{t}, N)] \tag{1.46}$$

Um jetzt eine Einschließung für $\varepsilon(\mathtt{t}, N)$ zu gewinnen, ist zu beachten, daß $g(r)$ eine gerade Funktion ist. Mit

$$r_0 := \max_{\zeta \in t}\{|r|\}, \quad r = \frac{\zeta - 1}{\zeta + 1} \quad \text{erhält man daher:}$$
$$\varepsilon(\mathtt{t}, N) \subseteq [0, g(r_0)]$$

Damit bleibt die Aufgabe, für $g(r_0)$ eine Oberschranke zu berechnen, die auf dem Rechner möglichst einfach auszuwerten ist. Wir betrachten dazu die Abschätzung:

$$\begin{aligned} g(r_0) &= \sum_{k=N+1}^{\infty} \frac{r_0^{2k}}{2k+1} = \frac{r_0^{2N+2}}{2N+3} \sum_{k=N+1}^{\infty} \frac{2N+3}{2k+1} \cdot r_0^{2k-(2N+2)} \\ &\leq \frac{r_0^{2N+2}}{2N+3} \sum_{k=0}^{\infty} r_0^{2k} = \frac{r_0^{2N+2}}{2N+3} \cdot \frac{1}{1-r_0^2} =: G(r_0, N) \end{aligned}$$

Mit der in r_0 monoton wachsenden Funktion $G(r_0, N)$ gilt damit:

$$\varepsilon(\mathtt{t}, N) \subseteq [0, G(r_0, N)]$$

Im Programm wird mit `xi.sup` eine Oberschranke für r_0 berechnet, und die einfache Intervallauswertung von $G(r_0, N)$ liefert mit dem neu berechneten Intervall `xi` die Einschließung $G(r_0, N) \in \mathtt{xi}$. Damit gilt dann

$$G(r_0, N) \leq \mathtt{xi.sup} \quad \rightsquigarrow \quad \varepsilon(\mathtt{t}, N) \subseteq [0, \mathtt{xi.sup}] \quad \rightsquigarrow$$
$$\log(\mathtt{t}) \subseteq \frac{2}{\ln(10)} \cdot \frac{t-1}{t+1} \cdot \big(P_N(\mathtt{v}) + [0, \mathtt{xi.sup}]\big) \tag{1.47}$$

Der Polynomgrad N

Nach (1.39) und (1.47) sind wir nun in der Lage, für jeden Polynomgrad N eine garantierte Einschließung von $\log(x)$ zu berechnen. Es bleibt jedoch noch das Problem zu lösen, zu gegebenem Präzisionsparameter $S =$ `Stagg_prec` den Polynomgrad N gerade so zu wählen, daß die $13 \cdot (S+1)$ Dezimalstellen der Einschließung von $\log(x)$ möglichst alle korrekt berechnet werden.

Schon an dieser Stelle ist jedoch ausdrücklich darauf hinzuweisen, daß der entsprechende Algorithmus zur Bestimmung von N aus Laufzeitgründen hinreichend vereinfacht werden kann. Sollte dadurch N um 1 zu groß oder zu klein ausfallen, so wird dadurch die korrekte Einschließung von $\log(x)$ überhaupt nicht beeinflußt, da durch die Auswertung von (1.39) und (1.47) das Intervall $\log(x)$ für **jedes** N garantiert eingeschlossen wird! Ein zu grob berechnetes N führt daher höchstens zu einer geringfügigen Überschätzung der Einschließung von $\log(x)$.

Zunächst wird festgelegt, wie die in (1.39) angegebene `integer`-Größe k in Abhängigkeit von $S =$ `Stagg_prec` zu wählen ist:

$$k := \begin{cases} 1\,, & \text{falls } 1 \le S \le 6, \\ 2\,, & \text{falls } 7 \le S \le 15, \\ 3\,, & \text{falls } 16 \le S \le 19 \end{cases}$$

Ein großer Präzisionsparameter, z.B. der Maximalwert $S = 19$, erfordert also die Berechnung der $128^3 = 2097152$-ten Wurzel aus dem Mantissen-Intervall m. Damit liegt die Einschließung t für $\sqrt[q]{m} \subseteq t$ sehr dich bei 1 und damit r aus (1.40) sehr dicht bei 0, so daß die dortige Potenzreihe sehr gut konvergieren wird und durch ein Taylorpolynom nicht zu großen Grades N in der durch S geforderten Genauigkeit gut approximiert werden kann.

Zur Berechnung von N wird zunächst eine ungefähre Oberschranke A des absoluten Fehlers bestimmt, wenn die Potenzreihe in (1.40) durch

$$\sum_{k=0}^{N} \frac{r^{2k}}{2k+1}, \quad r = \frac{\zeta - 1}{\zeta + 1}, \quad \zeta \in t$$

approximiert wird. Im Programm wählen wir $r = \mathtt{t}[1]$, wobei `t` durch die Anweisung `t := (t-1)/(t+1)` gegeben ist. Wegen $r = \mathtt{t}[1]$ benutzen wir lediglich ein reelles Element $r \in \mathtt{t}$, so daß sich die Berechnung des absoluten Fehlers nicht auf das ganze Intervall `t` beziehen wird, sondern nur auf das eine Element $r \in \mathtt{t}$. Auf die Berechnung von N wird sich diese Vereinfachung jedoch kaum auswirken! Für den Betrag des absoluten Fehlers gilt damit die

Abschätzung:

$$
\begin{aligned}
|\varepsilon_{abs}| &:= \frac{2p \cdot |r|}{\ln(10)} \sum_{k=N+1}^{\infty} \frac{r^{2k}}{2k+1} \\
&\leq \frac{2p \cdot |r|}{\ln(10)} \cdot \frac{r^{2N+2}}{2N+3} \cdot \frac{1}{1-r^2} \\
&\leq p \cdot \frac{|r|^{2N+3}}{2N+3} =: A
\end{aligned}
$$

Wegen $2/\ln(10) < 1$ und $r^2 \ll 1$ haben wir daher mit A einen recht einfachen Ausdruck für eine Oberschranke des absoluten Fehlers.

Zur Berechnung von N benötigen wir jetzt noch einen zweiten Ausdruck für den absoluten Fehler, der mit dem Präzisionsparameter S in Beziehung stehen muß. Mit einem ζ aus dem sehr schmalen Intervall x betrachten wir zunächst den Funktionswert $u := \log(\zeta)$. Wenn dann nach Vorgabe von S möglichst $13 \cdot (S+1)$ korrekte Dezimalstellen zu berechnen sind, so muß der Betrag des absoluten Fehlers ungefähr gegeben sein durch

$$|\varepsilon_{abs}| \approx |u| \cdot 10^{-13\cdot(S+1)}$$

Mit A verlangen wir daher die Ungleichung

$$p \cdot \frac{|r|^{2N+3}}{2N+3} < |u| \cdot 10^{-13\cdot(S+1)} \quad \Longleftrightarrow$$

$$\log(p) + (2N+3) \cdot \log(|r|) - \log(2N+3) < \log(|u|) - 13 \cdot (S+1) \tag{1.48}$$

Da im `real`–Format 10^{-267} die kleinste positive Zahl ist, macht es keinen Sinn, den absolute Fehler kleiner zu wählen als 10^{-267}. Wenn daher die rechte Seite von (1.48) kleiner wird als -267 oder wenn $|u|$ verschwindet, so wird diese rechte Seite auf den Wert -267 gesetzt. Mit $N = 1$ beginnend wird N dann in einer `repeat`–Schleife schrittweise um 1 erhöht, bis (1.48) erfüllt wird oder bis der Maximalwert $N = 25$ erreicht ist. Für $S = 19$ und $x = 2$ erhält man beispielsweise $N = 20$.

Anmerkung:

Um eine unnötige Intervallüberschätzung vor der Auswertung des Polynoms $P_N(v)$ aus (1.45) zu vermeiden, wird die Berechnung von $\sqrt[p]{m}$ und die Auswertung von $(t-1)/(t+1)$ in erhöhter Genauigkeit durchgeführt.

1.9.1.2 Der Programm-Quelltext

```
function log10(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := log10(x) liefert Einschliessung von log_10(x) }
const c = 1e-14;
   c1   = 128;     { 128^1 }
   c2   = 16384;   { 128^2 }
   c3   = 2097152; { 128^3 }
var t,v,erg        : Stagg[-1..Stagg_prec];
   xi              : interval;
   u,lgr,lgp,lgabs : real;
   ex,j,S,k,N      : integer;
   t1              : Stagg;
begin
   if Stagg_prec > 19 then s_fu1_math_error(2);
   t := x;  normal(t);
   if (t[0] -> t[-1])/>(t[0]+<t[1]) > c  then
   { Rechnung im einfachen real-Format: }
      t := Staggintval( log10(ival(t)) ) else
   begin { Jetzt schmales Intervallargument t: }
      if Stagg_prec = 1 then lgp := t[1] +> t[0]
                        else lgp := t[1] +> t[2] +> t[0];
      ex := expo(lgp);
      if -ex > 255 then
      begin
         t := t M_10HOCH 255;
         t := t M_10HOCH (-ex-255);
      end          else t := t M_10HOCH -ex;
      { log10(x) enthalten in:  ex + log10(t) }
      { Jetzt gilt ungefaehr:    t in [0.1,1] }
      if Stagg_prec = 1 then u := t[0] else u := t[2];
      if (ex=1) and (t[1]=0.1) and (u=0) and (t[-1]=t[0])
         then t := 0 else
      begin
         S := get_Stagg_prec;  k := 1;
         if S > 6 then
            if S < 16 then k := 2 else k := 3;
         allocate(t1,-1..S+1);  set_Stagg_prec(S+1);
         t1 := change_length(t);
```

```
        For j := 1 to k do t1 := sqrt128(t1);
        { Auswertung des Approximationspolynoms: }
        t1 := (t1-1) / (t1+1);
        set_Stagg_prec(S);
        t := change_length(t1);  free(t1);
        { Berechnung von N: }
        lgr := log10( abs(t[1]) );  lgabs := -13*(S+1);
        if lgp<>1 then
           lgabs := lgabs + log10( abs(log10(lgp)) )
                  else lgabs := -267;
        if lgabs < -267 then lgabs := -267;
        case k of
          1 : lgp := log10(c1);
          2 : lgp := log10(c2);
          3 : lgp := log10(c3);
        end; { case }
        N := 1;
        repeat
           N := N+1;
           j := 2 * N + 3;
        until (lgp+j*lgr-log10(j)<lgabs) or (N=25);
        v := t*t;   erg := 0;      { Auswertung von P_N(v): }
        For j := N downto 0 do
           erg := erg*v + Staggintval(1)/(2*j+1);
        { Beruecksichtigung des abs. Approximationsfehlers: }
        xi := abs( ival(t) );
        xi := intval(xi.sup);
        xi := power(xi,2*N+2)/( (2*N+3)*(1-xi*xi) );
        erg := erg + Staggintval(0,xi.sup);
        t := t * erg * change_length(q0);  { 2/ln(10) in q0 }
        case k of
          1 : u := c1;
          2 : u := c2;
          3 : u := c3;
        end; { case }
        t := t * u + ex;
     end;
  end;
  log10 := t;
end; { log10 }
```

Im vorhergehenden Quelltext ist die staggered Variable `q0` eine maximal-genaue Einschließung von $2/\ln(10) \in$ `q0`. Die mit Mathematica und Maple berechneten staggered Komponenten von `q0` lauten wie folgt:

```
{----------------------------------------------------------}
{         Konstanten fuer die Logarithmus-Funktion         }
{----------------------------------------------------------}
   { 2/ln(10) enthalten in q0[1]+..+q0[20] + [q0[-1],q0[0]] }
   q0[1]  := 8.685889638065e-1;
   q0[2]  := 3.655302257837e-15;
   q0[3]  := 8.332101645887e-28;
   q0[4]  := 9.401160733313e-41;
   q0[5]  := 2.228907566331e-54;
   q0[6]  := 7.292984177415e-67;
   q0[7]  := 4.945844989867e-80;
   q0[8]  := 6.863496637412e-93;
   q0[9]  := 2.134895326074e-106;
   q0[10] := 6.728335857431e-119;
   q0[11] := 7.927813138442e-132;
   q0[12] := 1.293256245317e-145;
   q0[13] := 4.254173137340e-159;
   q0[14] := 6.591867417393e-172;
   q0[15] := 1.765337666232e-185;
   q0[16] := 7.215476981028e-198;
   q0[17] := 5.688697333537e-211;
   q0[18] := 2.931721702711e-224;
   q0[19] := 2.296424697530e-237;
   q0[20] := 6.870868714634e-250;
   q0[-1] := 5.0767e-263;  q0[0] := 5.0768e-263;
```

1.9.1.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
{ y := log10(x) liefert Einschliessung von log_10(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := log10(x);
       writeln('log10(x)  enthalten in: ');
       write(y);
       writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert zum jeweils gegebenen staggered Intervallargument x für den Wertebereich $\log_{10}(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
10 0 0 0 0 0 0     <enter>
log10(x)  enthalten in:
 9.999999999999E-001
 9.999999999999E-014
 9.999999999999E-027
 9.999999999999E-040
 9.999999999999E-053
 9.999999999999E-066
[  9.999999999998E-079,  1.000000000001E-078 ]
```

```
Stagg_prec = ?
19     <enter>
x: Stagg   = ?
19 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       <enter>
log10(x)  enthalten in:
 3.010299956639E-001
 8.119521373889E-014
 4.724493026768E-027
 1.898814621085E-040
 4.131042746112E-053
 7.108189274424E-066
 5.094869272521E-079
 1.818617204068E-092
 4.477191430995E-105
 3.790947678811E-118
 3.352350599969E-131
 2.333704695575E-144
 6.450296425419E-158
 3.402661819734E-171
 3.116029435011E-184
 8.390289817858E-197
 2.617154439531E-210
 8.619290463538E-223
 8.469952023931E-236
[  8.496124625373E-250,  8.496124625499E-250 ]

Stagg_prec = ?
6       <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 1e-267     <enter>
log10(x)  enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -1.029670829252E-091,  1.083888823273E-091 ]
```

```
Stagg_prec = ?
6      <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0     <enter>
log10(x)  enthalten in:
-2.670000000000E+002
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -1.029670829252E-091,  1.083888823273E-091 ]

Stagg_prec = ?
19     <enter>
x: Stagg   = ?
19 Stagg components and 1 interval are needed in this sequence
22.3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
log10(x)  enthalten in:
 1.348304863048E+000
 1.606734751762E-013
 1.624035284445E-026
 3.424237980210E-039
 8.177231581520E-052
 9.268159188562E-065
 7.640665392658E-078
 6.104109753181E-091
 4.805773444582E-104
 9.106884984150E-117
 1.615942164837E-130
 2.632301399149E-143
 9.379860466828E-156
 2.546156682134E-169
 5.500342152739E-183
 6.471624113519E-196
 9.484552366939E-209
 7.523437641860E-222
 9.517179914433E-235
[  6.707906501159E-248,  6.707906501162E-248 ]
```

1.9.2 $\ln(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich $\ln(x)$. Wegen der Beziehung

$$\ln(x) \equiv \log_{10}(x) \cdot \ln(10)$$

erhält man mit $\log_{10}(x) \subseteq \mathtt{log10(x)}$ und $\ln(10) \subset \mathtt{ln10}$ diese Einschließung bei Anwendung der Inklusionsmonotonie durch

$$\ln(x) \subset \mathtt{log10(x)} \cdot \mathtt{ln10} \tag{1.49}$$

wobei die Multiplikation rechts in (1.49) als Multiplikation zweier staggered Intervalle zu verstehen ist. Der Quelltext der `ln`–Funktion ist damit gegeben durch

```
function ln(var x : Stagg) : Stagg[-1..Stagg_prec];
{ Logarithmus zur Basis e }
begin
   ln := log10(x) * change_length(ln10);
end; { ln }
```

Für ein gegebenes staggered Argument x gilt dann: $\ln(x) \subset \mathtt{ln(x)}$. Wenn $\inf(x) \leq 0$, so erfolgt eine entsprechende Fehlermeldung mit Programmabbruch. Das **PASCAL-XSC** Programm

```
program tests;
{ y := ln(x) liefert Einschliessung von log_e(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := ln(x);
       writeln('ln(x)  enthalten in: ');
       write(y);
       writeln;
    until false;
 end; { main }
```

```
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert zum jeweils gegebenen staggered Intervall-Argument x die folgenden Einschließungen von $\ln(x)$. Mit `set_Stagg_prec(p)` kann der Anwender den Präzisionsparameter `Stagg_prec` durch den `integer`–Wert `p` festlegen. Für die staggered Intervall-Arithmetik stehen dann $13 \cdot (p+1)$ Dezimalziffern zur Verfügung. Die `integer`–Variable `Stagg_prec` wird vom Modul `type_def` exportiert.

```
Stagg_prec = ?
19     <enter>
x: Stagg   = ?
19 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       <enter>
ln(x)  enthalten in:
 6.931471805599E-001
 4.530941723212E-014
 1.458176568075E-027
 5.001343602552E-040
 5.412068000949E-053
 3.393621969694E-066
 7.156058633269E-079
 9.641868754200E-092
 1.481020570685E-105
 7.336855202357E-118
 5.813055703267E-131
 7.516350759619E-145
 3.072757082837E-158
 1.435190307038E-171
 6.238916734711E-184
 2.335011536449E-197
 7.955239120475E-210
 1.726815749320E-223
 6.515552473413E-236
[  9.525882950444E-249,  9.525882950477E-249 ]
```

```
Stagg_prec = ?
19     <enter>
x: Stagg   = ?
19 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
ln(x)  enthalten in:
-6.147902198294E+002
-1.019763280371E-011
-8.400725243429E-024
-4.940974638823E-037
-8.460089854955E-050
-8.341886783853E-063
-1.122230112537E-076
-5.892274565730E-089
-5.398339290428E-102
-3.851203432991E-115
-7.710940402989E-128
-8.727549374711E-141
-9.193622462292E-155
-3.099785294571E-168
-9.186042640070E-181
-8.106591749375E-194
-5.854611901925E-207
-6.187813508154E-220
-2.470976132715E-233
[ -9.215665932957E-246, -9.215665932952E-246 ]

Stagg_prec = ?
6      <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
9.99e255 0 0 0 0 0 0    <enter>
ln(x)  enthalten in:
 5.894607833061E+002
 4.211157510566E-011
 9.416943168800E-024
 9.212258837154E-037
 4.745528095462E-050
 6.901342875016E-063
[  3.654657271616E-076,  3.654657271617E-076 ]
```

1.9.3 $\ln(1+x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich

$$\ln(1+x) := \{y \,|\, y = \ln(1+\zeta),\ \zeta \in x\}$$

Liegt x in der Nähe des Ursprungs, so liefert der interne Algorithmus der `ln(1+x)`–Funktion von Seite 61 deutliche Überschätzungen des Wertebereichs $\ln(1+x)$, so daß dann mit Hilfe der Potenzreihe von $\ln(1+\zeta)$ ein neuer Algorithmus zu entwickeln ist.

1.9.3.1 Der Algorithmus

Der Grobalgorithmus zur Einschließung von $\ln(1+x)$ lautet:

> `if` $x \subseteq [-10^{-8}, +10^{-8}]$ `then` Taylor-Approximation `else`
> `if Sup(`x`)` < 1 `then ln(1+x)` in erhöhter Prazision
> `else ln(1+x)` in `Stagg_prec`–Prazision auswerten.

Da die obigen Funktionsaufrufe `ln(1+x)` den Wertebereich von $\ln(1+x)$ garantiert einschließen, werden wir jetzt im Falle $x \subseteq [-10^{-8}, +10^{-8}]$ nur noch die Approximation durch eine abgebrochene Potenzreihe betrachten. Es gilt:

$$\begin{aligned}
\ln(1+x) &:= \left\{ y \,|\, y = \frac{\zeta}{2+\zeta} \sum_{k=0}^{\infty} \frac{2}{2k+1} \cdot r^k, \quad r := \left(\frac{\zeta}{2+\zeta}\right)^2,\ \zeta \in x \right\} \\
(1.50) \qquad &\subseteq \frac{x}{2+x} \left\{ y \,|\, y = \sum_{k=0}^{\infty} \frac{2}{2k+1} \cdot r^k, \quad \zeta \in x \right\} = \frac{x}{2+x} \cdot f(x) \\
\text{Mit } f_N(x) &:= \left\{ y \,|\, y = \sum_{k=0}^{N} \frac{2}{2k+1} \cdot r^k,\ \zeta \in x \right\} \text{wird } \varepsilon(x,N) \ \text{ definiert:}
\end{aligned}$$

$$f(x) = f_N(x) + \varepsilon(x,N), \tag{1.51}$$

und das Intervall des Approximationsfehlers $\varepsilon(x,N)$ ergibt sich daraus zu:

$$\varepsilon(x,N) := \left\{ y \,|\, y = \sum_{k=N+1}^{\infty} \frac{2}{2k+1} \cdot r^k, \quad r := \left(\frac{\zeta}{2+\zeta}\right)^2,\ \zeta \in x \right\}$$

Damit ist $\varepsilon(x,N)$ der Wertebereich der Funktion

$$g(r) := \sum_{k=N+1}^{\infty} \frac{2}{2k+1} \cdot r^k, \quad r \in R := \left\{ y \,|\, y = \left(\frac{\zeta}{2+\zeta} \right)^2, \; \zeta \in x \right\}$$

Da R im positiven Bereich liegt, gilt mit $r_0 := \sup(R)$ die Einschließung:

$\varepsilon(x,N) \subseteq [0, g(r_0)]$, und für $g(r_0)$ erhält man die Abschätzung:

$$\begin{aligned} g(r_0) &= \sum_{k=N+1}^{\infty} \frac{2}{2k+1} \cdot r_0^k = \frac{2 \cdot r_0^{N+1}}{2N+3} \sum_{k=N+1}^{\infty} \frac{2N+3}{2k+1} \cdot r_0^{k-(N+1)} \\ &\leq \frac{2 \cdot r_0^{N+1}}{2N+3} \sum_{j=0}^{\infty} r_0^j = \frac{2 \cdot r_0^{N+1}}{2N+3} \cdot \frac{1}{1-r_0} =: A \end{aligned} \tag{1.52}$$

Die Oberschranke A wird dann auf dem Rechner im einfachen `real`–Format intervallmäßig ausgewertet. Damit erreicht man, daß auf der Maschine mit `xi.sup` selbst wieder eine garantierte Oberschranke von A berechnet wird. Es ergibt sich damit für den Approximationsfehler die Einschließung:

$$\varepsilon(x,N) \subseteq [0, g(r_0)] \subseteq [0, A] \subseteq [0, \texttt{xi.sup}]$$

Beachte Sie bitte, daß $f_N(x)$ auf dem Rechner nicht direkt ausgewertet wird. Nach der Inklusionsmonotonie [2] gilt jedoch die Einschließung:

$$f_N(x) \subseteq P_N(x) := \sum_{k=0}^{N} \frac{2}{2k+1} \cdot t^k, \quad t := \left(\frac{x}{2+x} \right)^2$$

und genau dieses Polynom $P_N(x)$ wird in staggered Intervall-Arithmetik ausgewertet. Damit ergibt sich dann nach (1.50) und (1.51) für $\ln(1+x)$ die garantierte Einschließung:

$$\ln(1+x) \subseteq \frac{x}{2+x} \cdot \big(P_N(x) + [0, \texttt{xi.sup}] \big) \tag{1.53}$$

Beachten Sie bitte weiter, daß $x/(2+x)$ und $P_N(x)$ auf dem Rechner in staggered Intervall-Arithmetik ausgewertet werden, so daß für die genannten Intervalle auf der Maschine wieder automatisch garantierte Einschließungen berechnet werden. Damit wird die Einschließung von $\ln(1+x)$ in jedem Fall sichergestellt!

Die Berechnung des Polynomgrades N

Nach (1.53) sind wir nun in der Lage, für jeden Polynomgrad N eine garantierte Einschließung von $\ln(1+x)$ zu berechnen. Es bleibt jedoch noch das Problem zu lösen, zu gegebenem Präzisionsparameter $S =$ `Stagg_prec` den Polynomgrad N gerade so zu wählen, daß die $13 \cdot (S+1)$ Dezimalstellen der Einschließung von $\ln(1+x)$ möglichst alle korrekt berechnet werden.

Zur Berechnung von N wird zunächst eine ungefähre Oberschranke U des absoluten Fehlers bestimmt, wenn die Potenzreihe in (1.50) durch

$$\sum_{k=0}^{N} \frac{2}{2k+1} \cdot r^k, \quad r = \left(\frac{\zeta}{2+\zeta}\right)^2, \quad \zeta \in x$$

approximiert wird. Für $\zeta \in x \subseteq [-10^{-8}, +10^{-8}]$ wählen wir ein geeignetes ζ_0 und erhalten für r die gute Näherung $r \approx \zeta_0^2/4$. Nach (1.52) gilt dann für die ungefähre Oberschranke U des Betrags des absoluten Fehlers:

$$\frac{|\zeta_0|}{2+\zeta_0} \cdot \frac{2 \cdot r^{N+1}}{2N+3} \cdot \frac{1}{1-r} \approx \frac{|\zeta_0| \cdot r^{N+1}}{2N+3} =: U, \quad r \approx \zeta_0^2/4;$$

Zur Berechnung von N benötigen wir jetzt noch einen zweiten Ausdruck für den absoluten Fehler, der mit dem Präzisionsparameter S in Beziehung stehen muß. Mit einem ζ_0 aus dem sehr schmalen Intervall x betrachten wir zunächst den Funktionswert $\log(1+\zeta_0) \approx \zeta_0$. Wenn dann nach Vorgabe von S möglichst $13 \cdot (S+1)$ korrekte Dezimalstellen zu berechnen sind, so muß der Betrag des absoluten Fehlers ungefähr gegeben sein durch

$$|\varepsilon_{abs}| \approx |\zeta_0| \cdot 10^{-13 \cdot (S+1)}$$

Mit U verlangen wir daher die Ungleichung

$$\frac{|\zeta_0| \cdot r^{N+1}}{2N+3} < |\zeta_0| \cdot 10^{-13 \cdot (S+1)}, \quad r \approx \zeta_0^2/4; \tag{1.54}$$

Da im `real`–Format 10^{-267} die kleinste positive Zahl ist, macht es keinen Sinn, den absolute Fehler kleiner zu wählen als 10^{-267}. Wenn daher die rechte Seite von (1.54) kleiner wird als 10^{-267}, so wird diese rechte Seite auf den Wert 10^{-267} gesetzt:

$$\frac{|\zeta_0| \cdot r^{N+1}}{2N+3} < 10^{-267}, \quad r \approx \zeta_0^2/4; \tag{1.55}$$

In logarithmischer Schreibweise lauten die Ungleichungen (1.54) und (1.55) mit $v := [2 \cdot \log(|\zeta_0|) - \log(4)]$ wie folgt:

$$(N+1)\cdot v - \log(2N+3) \quad < \quad 13\cdot(S+1) \tag{1.56}$$

$$(N+1)\cdot v - \log(2N+3) \quad < \quad -267 - \log(|\zeta_0|), \quad \zeta_0 \in x \tag{1.57}$$

Mit $N = 1$ beginnend wird N dann in einer `repeat`–Schleife schrittweise um 1 erhöht, bis (1.56) bzw. (1.57) erfüllt wird. Für $S = 18$ und $x = 9.9\cdot 10^{-9}$ erhält man beispielsweise $N = 14$. Um $|\zeta_0| = 0$ zu verhindern, wird $|\zeta_0|$ wie folgt berechnet:

$$|\zeta_0| := |x[1]| \;+\!\!> |x[2]| \;+\!\!> \max\{\,|x[-1]|, |x[0]|\,\}$$

Wegen $0 \notin x$ kann die obige rechte Seite nicht verschwinden. Es ist jedoch möglich, daß $|\zeta_0|$ geringfügig größer ausfällt als $\mathrm{Sup}(|x|)$. Dies ist jedoch kein Unglück, denn bei der Berechnung von N nach (1.56) oder (1.57) kommt es nicht sosehr auf den speziellen Wert sondern auf die Größenordnung von ζ_0 an. Aber selbst wenn N um 1 zu groß oder zu klein berechnet wurde, hat dies auf die korrekte Einschließung von $\ln(1+x)$ keinerlei Einfluß, da (1.53) für **jedes** N gilt; vergleichen Sie dazu auch die entsprechenden Bemerkungen von Seite 53.

1.9.3.2 Der Programm-Quelltext

```
function ln1p(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := ln1p(x) liefert Einschliessung von ln(1+x) }
const c = 1e-8;
var t,zeta,z2 : Stagg[-1..Stagg_prec];
   t1         : Stagg;
   u,v        : real;
   xi         : interval;
   S,N,p      : integer;
begin
   if Stagg_prec > 18 then s_fu1_math_error(9);
   t := x;  Normal(t);  S := Stagg_prec;
   if S = 1 then
   begin
      u := t[-1];   v := t[0]
   end      else
   begin
      u := t[2] +< t[-1];
```

```
      v := t[2] +> t[0];
   end;
   if (t[1] +< u > -c) and (t[1] +> v < c) then
      if (t[1]=0) and (t[-1]=t[0]) then t := 0   { x = 0 }
      else
      begin
         { Bestimmung von N: }
         v := abs(t[-1]);  u := abs(t[0]);
         if u > v then v := u;
         if S > 1 then v := abs(t[2]) +> v;
         v := log10( abs(t[1]) +> v);
         { v: reelle log(|x|)-Naeherung im real-Format }
         p := 13 * (S+1);
         if v-p < -267 then u := -267 - v else u := -p;
         v := 2*v - log10(4);  N := 0;
         repeat
            N := N+1;
         until (N+1)*v - log10(2*N+3) < u; { N berechnet! }
         zeta := t/(2+t);
         z2 := zeta*zeta;  xi := ival(z2);
         t := change_length(a_ln1p[N]);
         for j := N-1 downto 0 do
            t := t*z2 + change_length(a_ln1p[j]);
         { Die Beruecksichtigung des absoluten Fehlers: }
         xi := intval(xi.sup);
         xi := 2*power(xi,N+1)/((2*N+3)*(1-xi));
         t := t + Staggintval(0,xi.sup); { ist jetzt erfolgt }
         t := zeta * t;
      end                           else
   begin
      if t[1] +> v < 1 then
      begin { Rechnung in erhoehter Genauigkeit: }
         allocate(t1,-1..S+1);  set_Stagg_prec(S+1);
         t1 := 1 + change_length(t);
         t1 := ln(t1);   set_Stagg_prec(S);
         t := change_length(t1);  free(t1);
      end          else t := ln(1+t); { Einfache Genauigkeit }
   end;
   ln1p := t;
end; { ln1p }
```

1.9.3.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
{ y := ln1p(x) liefert Einschliessung von ln(1+x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := ln1p(x);
       writeln('ln(1+x)  enthalten in: ');
       write(y);
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\ln(1+x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1e-10 0 0 0 0 0 0     <enter>
ln(1+x)  enthalten in:
 9.999999999500E-011
 3.333333333083E-031
 3.333333533333E-044
 3.333166666666E-057
 6.809523809511E-070
 3.095238106349E-083
[  2.063482063492E-096,  2.063482063493E-096 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg   = ?
18 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   <enter>
ln(1+x)  enthalten in:
 9.999999999999E-021
 9.999999500000E-034
 3.333333333333E-061
 3.333333083333E-074
 3.333333333333E-087
 3.533333333333E-100
 3.333333316666E-113
 6.666666666666E-126
 6.680952380952E-139
 3.809523808273E-152
 8.095238095238E-165
 9.534920634920E-179
 6.349206348206E-192
 3.492063492063E-205
 4.921544011544E-218
 1.154401153567E-232
 8.210678210678E-245
 0.000000000000E+000
[  2.107551335000E-258,  2.107551339000E-258 ]

Stagg_prec = ?
7      <enter>
x: Stagg   = ?
7 Stagg components and 1 interval are needed in this sequence
1e-170 0 0 0 0 0 0 0   <enter>
ln(1+x)  enthalten in:
 9.999999999999E-171
 9.999999999999E-184
 9.999999999999E-197
 9.999999999999E-210
 9.999999999999E-223
 9.999999999999E-236
 9.999999999999E-249
[  9.999960000000E-262,  1.000002000000E-261 ]
```

1.9.4 $\log_a(x)$

Zu gegebenen staggered Argumenten $a = a_1 + a_2 + \ldots + a_{\mathrm{ub}(a)} + [a_{-1}, a_0]$ und $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich

$$\begin{aligned}
\log_a(x) &:= \left\{y \,|\, y = \log_\alpha(\zeta);\ \alpha \in a,\ 1 \notin a,\ \inf(a) > 0,\ \zeta \in x,\ \inf(x) > 0\right\} \\
&= \left\{y \,|\, y = \frac{\log_{10}(\zeta)}{\log_{10}(\alpha)};\ \alpha \in a,\ 1 \notin a,\ \inf(a) > 0,\ \zeta \in x,\ \inf(x) > 0\right\} \\
&= \frac{\log_{10}(x)}{\log_{10}(a)} = \frac{\ln(x)}{\ln(a)}, \qquad 1 \notin a,\ \inf(a) > 0,\ \inf(x) > 0; \qquad (1.58)
\end{aligned}$$

Mit Hilfe der schon implementierten Funktionen `log10` und `ln` gilt dann

$$\log_a(x) \subseteq \frac{\mathtt{log10(x)}}{\mathtt{log10(a)}}, \quad 1 \notin a,\ \inf(a) > 0,\ \inf(x) > 0, \text{ bzw.} \qquad (1.59)$$

$$\log_a(x) \subseteq \frac{\mathtt{ln(x)}}{\mathtt{ln(a)}}, \quad 1 \notin a,\ \inf(a) > 0,\ \inf(x) > 0; \qquad (1.60)$$

Gilt beispielsweise $x = 1 + \eta$ mit $\eta \subset [-\varepsilon, +\varepsilon], \varepsilon \ll 1$, so empfiehlt sich die Auswertung von `ln1p`(η) an Stelle von `ln(x)`, da der erste Ausdruck eine höhere Genaigkeit liefert. Im Programm wird die entsprechende Abfrage realisiert durch[3]:

```
if x[1] = 1 then ....
```

und $\eta = x - 1$ ergibt sich dann durch die Anweisung `x[1] := 0`. Für das Intervall-Argument a gelten die ganz analogen Überlegungen. Weitere Einzelheiten kann man dem folgenden, leicht lesbaren Quelltext entnehmen.

```
function loga(var a,x : Stagg) : Stagg[-1..Stagg_prec];
{ Logarithmus zur Basis a;    1 not in a, inf(x),inf(a) > 0 }
var aa,xx  : Stagg[-1..Stagg_prec];
   bla,blx : boolean;
begin
   aa := a;  Normal(aa);  bla := aa[1] = 1;
   xx := x;  Normal(xx);  blx := xx[1] = 1;
   if bla or blx then
   begin
      if bla then
      begin aa[1] := 0;  aa := ln1p(aa) end
```

[3]Da sich die $x[j]$ nicht überlappen, gilt: $|x[2]| \leq 10^{-13}$ und daher $\varepsilon \ll 1$.

```
      else aa := ln1p(aa-1);
      if blx then
      begin xx[1] := 0;  xx := ln1p(xx) end
      else xx := ln1p(xx-1);
      loga := xx / aa;
   end           else loga := log10(x) / log10(a);
end; { loga }
```

Beachten Sie bitte, daß sich die von Null verschiedenen `aa[j]` nach dem Prozeduraufruf `Normal(aa)` nicht überlappen! Entsprechendes gilt für die von Null verschiedenen `xx[j]`.

Numerische Ergebnisse:

Das **PASCAL-XSC** Programm

```
program tests;
{ y := loga(a,x) liefert Einschliessung von log_a(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var a,x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('a: Stagg   = ?');   read(a);
       writeln('x: Stagg   = ?');   read(x);
       y := loga(a,x);
       writeln('log_a(x)  enthalten in: ');
       write(y);
       writeln;
    until false;
 end; { main }
begin {tests }
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\log_a(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
2       <enter>
a: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
1 0 1e-260   <enter>
x: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
1 0 1e-20    <enter>
log_a(x)  enthalten in:
 0.000000000000E+000
 0.000000000000E+000
[  9.999991999980E+239,  1.000001000000E+240 ]
```

Es ist zunächst erstaunlich, daß unser Ergebnis nur auf 6 Dezimalstellen genau berechnet wird, obwohl mit `Stagg_prec = 2` eine Präzision von 39 Dezimalziffern vorgegeben wurde. Diese geringe Genauigkeit wird aber verständlich, wenn man beachtet, daß $\ln(1 + 10^{-260}) \approx 10^{-260}$ nur auf etwa 6 Dezimalstellen genau berechnet werden kann, da im `real`–Format 10^{-267} die kleinste darstellbare, positive Zahl ist! Mit diesem ungenauen Zwischenergebnis kann man dann natürlich nicht erwarten, daß der zu berechnende Quotient $\ln(1+10^{-20})/\ln(1+10^{-260})$ in größerer Genauigkeit geliefert wird.

```
Stagg_prec = ?
6       <enter>
a: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1 1e-160 0 0 0 0 0   <enter>
x: Stagg   = ?
6 Stagg components and 1 interval are needed in this sequence
1 1e-15 0 0 0 0 0    <enter>
log_a(x)  enthalten in:
 9.999999999999E+144
 9.950000000000E+131
 3.333333333333E+114
 3.308333333333E+101
 3.333533333333E+088
 3.333331666666E+075
[  6.666666680727E+062,  6.666666680958E+062 ]
```

```
Stagg_prec = ?
18     <enter>
a: Stagg   = ?
18 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
x: Stagg   = ?
18 Stagg components and 1 interval are needed in this sequence
8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
log_a(x)  enthalten in:
 2.999999999999E+000
 9.999999999999E-013
 9.999999999999E-026
 9.999999999999E-039
 9.999999999999E-052
 9.999999999999E-065
 9.999999999999E-078
 9.999999999999E-091
 9.999999999999E-104
 9.999999999999E-117
 9.999999999999E-130
 9.999999999999E-143
 9.999999999999E-156
 9.999999999999E-169
 9.999999999999E-182
 9.999999999999E-195
 9.999999999999E-208
 9.999999999999E-221
[  9.999999999999E-234,  1.000000000001E-233 ]

Stagg_prec = ?
1
a: Stagg   = ?
1 Stagg components and 1 interval are needed in this sequence
0 [1.000000000001,9.99e255]
x: Stagg   = ?
1 Stagg components and 1 interval are needed in this sequence
1 0
log_a(x)  enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

1.10 Die Exponentialfunktionen

Für eine BCD–Version empfiehlt sich, zunächst die Exponentialfunktion zur Basis 10 zu berechnen, da man für den ganzzahligen Anteil n des Exponenten x das Ergebnis 10^n im BCD–Format ohne Rechnung direkt angeben kann[4]. Die Exponentialfunktion zur Basis 10 muß dann nur noch für den restlichen Exponenten $(x - n)$ berechnet werden, der betragsmäßig kleiner als $1/2$ ist. Die Multiplikation mit 10^n erfolgt mit dem schnellen Operator `M_10HOCH`. Ist 10^x bekannt, so können die Funktionen e^x und 2^x mit Hilfe der Beziehungen

$$e^x = 10^{x \cdot \log_{10}(e)}, \qquad 2^x = 10^{x \cdot \log_{10}(2)}$$

realisiert werden, wobei die Konstanten $\log_{10}(e)$ und $\log_{10}(2)$ im Modul `s_fu1` als maximalgenaue Staggered–Intervall–Konstanten zur Verfügung stehen.

1.10.1 10^x

Zu berechnen ist eine Einschließung der Funktion 10^x, wobei x ein staggered Intervall ist, dessen Präzision durch die Variable `Stagg_prec` im Hilfsmodul `type_def` festgelegt ist und mit Hilfe der Funktion `set_Stagg_prec(p)` auf den `integer`–Wert `p` geändert werden kann.

Im ersten Schritt wird im Fall $x = 0$ das Ergebnis auf das Punktintervall 1 gesetzt, da $10^0 = 1$ ist.

1.10.1.1 Zu breite Exponenten-Intervalle

In der numerischen Praxis werden die Exponenten-Intervalle x meist extrem schmale Intervalldurchmesser haben oder sogar nur punktförmig sein. Ist jedoch x das Ergebnis vorhergehender staggered Intervall–Rechnungen, so kann nicht ausgeschlossen werden, daß diese Intervalle durch die bekannten Intervallüberschätzungen sehr breit ausfallen und dabei möglicherweise auch die Zahl Null enthalten. Für die Behandlung solcher zu breiten Intervalle x gibt es prinzipiell zwei Möglichkeiten:

1. Wegen der Monotonie der Funktion 10^x berechnet man in getrennten Rechnungen Einschließungen für $y_1 = 10^{\inf(x)}$ und $y_2 = 10^{\sup(x)}$. Das Intervall 10^x wird dann eingeschlossen durch die konvexe Hülle von y_1 und y_2, wobei $\inf(x)$ und $\sup(x)$ staggered Punktintervalle sind.

[4] n ist der mit `round(...)` gerundete Mittelpunkt des Argument-Intervalls x

2. Wenn der relative Durchmesser von x größer ist als 10^{-15}, wird 10^x mit Hilfe der Funktion `exp10(ti)` nur im einfachen `real`–Format eingeschlossen, wobei das Intervall `ti` durch `ti := ival(x)` eine Intervall-einschließung der staggered Größe `x` im `real`–Format ist. Andernfalls, d.h. also bei hinreichend schmalen Intervallen x, wird der Algorithmus zur Einschließung von 10^x nur mit dem einen Intervall x durchlaufen, dessen Durchmesser sehr schmal ist aber durchaus von Null verschieden sein kann.

Beim Auftreten solcher zu breiter Intervalle wählen wir aus Laufzeitgründen die Vorgehensweise nach Punkt 2, die im Programm durch den Sprung zum Label 1 realisiert wird. Dieser Sprung zum Label 1 erfolgt noch in den beiden folgenden Fällen:

a. Der Intervalldurchmesser von x ist kleiner als 10^{-15} und der ganzzahlige Anteil von x ist kleiner als -267, d.h. das Intervall x liegt im denormalisierten `real`–Bereich.

b. Die Auswertung des Algorithmus zur Einschließung von 10^x liefert ein Intervall, das wegen interner Intervallüberschätzungen einen Intervall-durchmesser liefert, der größer ist als 10^{-15}. In diesem Fall vermeidet man unnötige Intervallüberschätzungen, da die Funktion `exp10(ti)` im einfachen `real`–Format für beliebige `real`–Intervalle `ti` hochgenaue Einschließungen liefert.

1.10.1.2 Vorbereitungen

Um die Eingangsgröße x im Algorithmus nicht zu verändern, wird durch die Anweisung `t := x` das Argument x zunächst nach t kopiert; `normal(t)` sorgt dafür, daß sich die $t[j]$ nicht überlappen. Im Fall $t = 0$ wird das Ergebnis ohne Rechnung auf 1 gesetzt, da $10^0 = 1$ ist. Der Programmteil

```
kurz := false;
if d > 0 then
begin
   if 0 in t then kurz := d > c2
   else
      begin
         TRANS(t,u1);
         kurz := d/abs(u1) > c2;
      end;
end;
```

setzt die Boolesche Variable `kurz` auf `true`, falls $0 \in x$ und $\texttt{diam}(x) > 10^{-15}$ erfüllt ist oder wenn der relative Durchmesser von x größer ist als 10^{-15}. Im Fall `kurz = true` wird dann durch die Anweisungen

```
                  { var res : Stagg[-1..Stagg_prec]; }
ti := ival(t);    { var ti : interval;                }
res := Staggintval(exp10(ti));
```

bei zu breitem Argument–Intervall t das Intervall 10^t nur im einfachen `real`–Format durch `exp10(ti)` eingeschlossen und dann mit der Funktion `Staggintval(...)` in die staggered Variable `res` rücktransformiert.

Im Fall `kurz = false` ist der relative Intervalldurchmesser kleiner oder gleich 10^{-15}. Mit den Programm–Anweisungen

```
if t[1] < -1e8 then kurz := true
else
begin
   expo := round(t[1]); { expo: Gerundeter ganzzahl. }
   kurz := expo < -267; {       Exponent.            }
end;
if kurz then goto 1;
```

erfolgt über den Sprung zum Label 1 die Einschließung von 10^t wieder nur im einfachen `real`–Format, wenn der Zehnerexponent `expo` von $t[1]$ kleiner ist als -267, d.h. wenn 10^t im positiven, denormalisierten `real`–Bereich liegt. Andernfalls gilt jetzt:

$$10^t = 10^{expo} \cdot 10^{t-expo},$$

und mit `t := t-expo` bleibt die Einschließung von 10^t zu realisieren, wobei der relative Durchmesser von t kleiner ist als 10^{-15}. Um die einfache Taylorreihe der Exponentialfunktion mit der Basis e anwenden zu können, benutzen wir die Beziehung

$$10^t = e^{t \cdot \ln(10)}$$

und berechnen mit der Anweisung `t := t * change_length(ln10)` eine Einschließung von $t \cdot \ln(10)$.

Zusammenfasung: Für `t : Stagg[-1..Stagg_prec]` ist einzuschließen e^t, dabei gilt: $10^x \subseteq 10^n \cdot e^t$, der relativer Durchmesser von t ist kleiner oder gleich 10^{-15} und $t \subseteq [-1.152, +1.152]$, wobei die beiden letzten Beziehungen im nachfolgenden Algorithmus nicht direkt verwendet werden.

1.10.1.3 Argumentreduktion

Einzuschließen ist also das Intervall e^t, mit $t \subseteq T := [-1.152, +1.152]$. Dabei ist T natürlich viel zu breit, um e^t mit vertretbarem Aufwandt durch seine Taylorreihe auf z.B. 200 Dezimalstellen genau zu approximieren. Mit Hilfe der Beziehung

$$e^t = \left(e^{\frac{t}{10^k}}\right)^{10^k}, \qquad k = 1, 2, \ldots \tag{1.61}$$

läßt sich das Intervall t auf ein hinreichend schmales Intervall $\frac{t}{10^k} = t \cdot 10^{-k}$ reduzieren, wenn man k nur hinreichend groß wählt. Die Multiplikation mit 10^{-k} erfolgt mit dem schnellen Operator `M_10HOCH`, und die später notwendige Potenzierung mit 10^k erfolgt mit der Funktion `power(f1,k)`. Für den Maximalwert $k = 6$ werden dabei nur 26 Multiplikationen benötigt!

Natürlich wird man bei größeren `Stagg_prec`–Werten auch entsprechend schmale Approximationsintervalle, d.h. größere k-Werte wählen, um zu große Polynomgrade N zu vermeiden. Setzt man $\zeta_t := \sup(\mathrm{abs}(t))$ und verlangt mit $c1 := 0.17$ durch Wahl eines hinreichend großen k die Ungleichung

$$\zeta_t \cdot 10^{-k} < 10^{-\texttt{Stagg\_prec} \cdot c1}, \tag{1.62}$$

so gilt für das reduzierte Intervall $t \cdot 10^{-k}$ die Aussage:

$$t \cdot 10^{-k} \subset \left[-10^{-\texttt{Stagg\_prec} \cdot c1}, +10^{-\texttt{Stagg\_prec} \cdot c1}\right],$$

d.h. das reduzierte Intervall liegt in einem hinreichend schmalen Intervall. Die Gleichung (1.62) ist äquivalent mit

$$k > c1 \cdot \texttt{Stagg\_prec} + \log_{10}(\zeta_t)$$

und wird im Programm realisiert durch:

```
res := sup( abs(t) ) + 0;
k := trunc(c1*Stagg_prec + log10(res[1]));
```

Die Konstante $c1 = 0.17$ ist ein reiner Erfahrungswert. Die Einschließung des reduzierten Intervalls $t \cdot 10^{-k}$ erfolgt dabei im Programm mit Hilfe der dort vereinbarten staggered Variablen `t`, wobei diese Variable `t` durch die Anweisung `t := t M_10HOCH -k` realisiert wird.

1.10.1.4 Approximation von e^t

Die in (1.61) in Klammern stehende Exponentialfunktion wird jetzt nach Ausführung der Anweisung `t := t M_10HOCH -k` mit e^t bezeichnet und ist durch ihre abgebrochene Taylorreihe vom Grade N zu approximieren:

$$e^t \approx \sum_{n=0}^{N} \frac{t^n}{n!} \tag{1.63}$$

Für das reduzierte Intervall t gilt: $t \subset \left[-10^{-\texttt{Stagg\_prec}\cdot c1}, +10^{-\texttt{Stagg\_prec}\cdot c1}\right]$, wobei der relative Durchmesser von t kleiner 10^{-15} ist und das Intervall t auch die Zahl Null enthalten kann. Wählt man z.B. `Stagg_prec = 3` und das Argument $x = 20 + 0 + 0 + [-10^{-30}, +10^{-30}]$, so liefert der beschriebene Algorithmus das reduzierte staggered Argument

$$t = 0 + 0 + 0 + [-2.302585092995 \cdot 10^{-30}, +2.302585092995 \cdot 10^{-30}],$$

welches die Null enthält.

Abschätzung des Approximationsfehlers

Zu gegebenem Polynomgrad N soll jetzt nach (1.63) der absolute Approximationsfehler abgeschätzt werden. Dazu nehmen wir an, daß t zunächst keine negativen Elemente enthalten soll. Mit $\varepsilon > 0$ sei damit t gegeben durch $t := [0, \varepsilon]$. Für $\zeta \in t$ gilt dann die folgende Abschätzung:

$$\begin{aligned}
\left| e^\zeta - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right| &= \sum_{n=N+1}^{\infty} \frac{\zeta^n}{n!} \\
&= \frac{\zeta^{N+1}}{(N+1)!} \sum_{n=N+1}^{\infty} \frac{(N+1)!}{n!} \zeta^{n-(N+1)} \\
&\leq \frac{\zeta^{N+1}}{(N+1)!} \sum_{n=0}^{\infty} \zeta^n = \frac{\zeta^{N+1}}{(N+1)!} \cdot \frac{1}{1-\zeta} =: \varepsilon_{abs}
\end{aligned}$$

Da ε_{abs} für $0 \leq \zeta < 1$ monoton wächst, erhält man das Ergebnis:

$$\left| e^\zeta - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right| \leq \frac{\varepsilon^{N+1}}{(N+1)!} \cdot \frac{1}{1-\varepsilon}, \qquad \zeta \in t = [0, \varepsilon]; \tag{1.64}$$

Jetzt wird der absolute Approximationsfehler abgeschätzt, wenn das Intervall t keine positiven Werte enthält, d.h. t sei gegeben durch $t := [-\varepsilon, 0]$. Für $\zeta \in t$ gilt dann die folgende Abschätzung:

$$\begin{aligned}\left| e^\zeta - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right| &= \left| \sum_{n=N+1}^{\infty} \frac{\zeta^n}{n!} \right| \leq \sum_{n=N+1}^{\infty} \frac{|\zeta^n|}{n!} = \sum_{n=N+1}^{\infty} \frac{|\zeta|^n}{n!} \\ &\leq \sum_{n=N+1}^{\infty} \frac{\varepsilon^n}{n!} \leq \frac{\varepsilon^{N+1}}{(N+1)!} \cdot \frac{1}{1-\varepsilon}, \qquad \zeta \in [-\varepsilon, 0]\end{aligned}$$

Man erhält also für die Oberschranke des absoluten Approximationsfehlers das gleiche Ergebnis (1.64) wie bei positiven ζ-Werten!

Ist t jetzt ein beliebiges reduziertes Intervall, das positive und negative Werte enthalten kann und ist ζ_t das Maximum der Beträge seiner Elemente, so gilt mit dem obigen Ergebnis für den absoluten Approximationsfehler die Abschätzung:

$$\left| e^\zeta - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right| \leq \frac{\zeta_t^{N+1}}{(N+1)!} \cdot \frac{1}{1-\zeta_t}, \quad \zeta \in t, \quad \zeta_t := \sup(\text{abs}(t)); \tag{1.65}$$

Um eine garantierte Oberschranke des absoluten Approximationsfehlers zu erhalten, muß die rechte Seite von (1.65) intervallmäßig ausgewertet werden. Aus Laufzeitgründen wird man diese Auswertung jedoch nicht im staggered Format sondern im einfachen `interval`–Format vornehmen. Die Funktion `ival(t)` liefert dabei zunächst eine Einschließung des staggered Intervalls t durch ein Intervall vom Typ `interval`. Mit den Programm–Anweisungen:

```
ti := sup( abs(ival(t)) );     { ti : interval }
ti := power(ti,N+1) / (factorial_h(N+1)*(1-ti));
```

liefert dann das Intervall $[-\texttt{ti.sup}, +\texttt{ti.sup}]$ eine garantierte Einschließung des absoluten Approximationsfehlers.

Anmerkungen:
Die Intervallrechnung garantiert nach den beiden obigen Anweisungen:

- $\zeta_t < \sup(\texttt{abs}(\texttt{ival}(\texttt{t})))$,
- $\dfrac{\zeta_t^{N+1}}{(N+1)!} \cdot \dfrac{1}{1-\zeta_t} < \texttt{ti.sup}$,

d.h. $[-\texttt{ti.sup}, +\texttt{ti.sup}]$ ist tatsächlich eine garantierte Einschließung des absoluten Approximationsfehlers.

1.10.1.5 Der Polynomgrad N

Der für einen vogegebenen `Stagg_prec`–Wert S notwendige Polynomgrad N ergibt sich aus der Forderung

$$\frac{\left| e^{\zeta} - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right|}{e^{\zeta}} < 10^{1-13\cdot(S+1)}, \qquad \zeta \in t$$

Man verlangt damit, daß der Betrag des relativen Approximationsfehlers so klein gewählt wird, daß alle $13\cdot(S+1)$ Dezimalziffern des staggered Formates richtig berechnet werden. Diese Idealforderung kann in der Praxis natürlich nicht vollständig umgesetzt werden, da wegen der unvermeidbaren Intervallüberschätzungen immer einige wenige Ziffern ungenau werden müssen.

Das reduzierte Intervall t wurde durch Multiplikation des ursprünglichen Intervalls mit $10^{-k}, k = 1, 2, \ldots$ gewonnen, so daß $\zeta \in [-0.1152, +0.1152]$ nach Seite 78 oben gesichert ist. Damit läßt sich dann obige Ungleichung zur Bestimmung von N vereinfachen zu:

$$\left| e^{\zeta} - \sum_{n=0}^{N} \frac{\zeta^n}{n!} \right| < 10^{1-13\cdot(S+1)}, \qquad \zeta \in t,$$

und mit (1.65) erhält man:

$$\frac{\zeta_t^{N+1}}{(N+1)!} \cdot \frac{1}{1-\zeta_t} < 10^{1-13\cdot(S+1)}, \qquad \zeta_t := \sup(\mathrm{abs}(t)) < 0.1152,$$

und dies läßt sich nochmals vereinfachen zu:

$$\frac{\zeta_t^{N+1}}{(N+1)!} < 10^{1-13\cdot(S+1)}, \qquad \zeta_t := \sup(\mathrm{abs}(t)) < 0.1152, \tag{1.66}$$

Beachten Sie bitte, daß die obigen Vereinfachungen durchaus zulässig sind und eine garantierten Einschließung von 10^x keinesfalls verhindern! Sollte in einem Sonderfall N tatsächlich um 1 zu klein berechnet werden, so muß nur dafür gesorgt werden, daß der absolute Approximationsfehler nach (1.65) richtig eingeschlossen und im folgenden Algorithmus richtig berücksichtigt wird. Ein zu kleines N erzeugt dann lediglich eine etwas größere Intervallüberschätzung bei der Einschließung von 10^x. Ein zu kleiner Wert von N ist jedoch sehr unwahrscheinlich, da man, mit $N = 1$ beginnend, N solange in Dreierschritten vergrößert, bis (1.66) erfüllt wird.

Im Programm wird man aus Laufzeitgründen nicht (1.66) sondern die logarithmierte Ungleichung

$$1 - 13 \cdot (S+1) + \log_{10}(N+1)! > (N+1) \cdot \log_{10}(\zeta_t) \tag{1.67}$$

in Dreierschritten im `real`–Format auswerten, wobei $1 - 13 \cdot (S+1)$ und $\log_{10}(\zeta_t)$ in `real`–Variablen gespeichert werden. Im Programm erfolgt die Berechnung von N durch die folgenden Anweisungen:

```
u1 := 1 - 13*(Stagg_prec+1);
u3 := sup( abs(ival(t)) );  u2 := log10(u3);
N := 1;  d := 1;
repeat
   d := d*(N+1)*(N+2)*(N+3);
   N := N + 3;
until u1 + log10(d*(N+1)) > (N+1)*u2;
```

Die Anweisung `d := d*(N+1)*(N+2)*(N+3)` verhindert, daß $(N+1)!$ in jedem Dreierschritt zeitaufwendig neu berechnet werden muß.

1.10.1.6 Einschließung von e^t

Die Approximation von e^t erfolgt mit der abgebrochenen Taylorreihe:

$$e^t \approx P_N(t) := \sum_{n=0}^{N} \frac{t^n}{n!},$$

wobei $P_N(t)$ nach folgendem Schema ausgewertet wird:

$$P_N(t) = \left(\ldots \left(\left(\frac{t}{N} + 1 \right) \frac{t}{N-1} + 1 \right) \ldots + 1 \right) \frac{t}{2} + 1$$

Ist nun `PN` das Ergebnis der staggered Intervallauswertung von $P_N(t)$, so gilt zunächst $P_N(t) \subseteq \texttt{PN}$. Ist außerdem das Intervall $[-\texttt{ti.sup}, +\texttt{ti.sup}]$ nach Seite 80 eine Einschließung des absoluten Approximationsfehlers, so gilt:

$$e^t \subseteq \texttt{PN} + [-\texttt{ti.sup}, +\texttt{ti.sup}],$$

so daß mit der obigen rechten Seite eine garantierte Einschließung von e^t berechnet wird. Im Programm erfogt die Einschließung von e^t durch `PN` mit Hilfe der folgenden Anweisungen:

```
PN := t / N;
for j := N-1 downto 1 do PN := ( PN+1 )*t / j;
PN := PN + 1 + Staggintval(-ti.sup,ti.sup); { e^t in PN }
```

1.10.1.7 Zurücksetzen der Argumentreduktion

Bis jetzt haben wir für den in (1.61) geklammerten Ausdruck, den wir in den letzten Abschnitten mit e^t bezeichnet haben, mit PN eine garantierte Einschließung berechnet. Nach (1.61) muß dieses PN noch mit 10^k potenziert werden, um das ursprüngliche Intervall e^t links in (1.61) einschließen zu können. Im Programm erfolgt dies durch die Anweisungen:

```
k := trunc( comp(0.1,k+1) );
PN := power(PN,k);
res := PN M_10HOCH expo;    { Multiplik. mit 10^expo }
```

wobei im letzten Schritt die Abspaltung des ganzzahligen Exponentenanteils rückgängig gemacht wird, so daß `res` die gesuchte garantierte Einschließung von 10^x ist.

1.10.1.8 Zu breite Ergebnisintervalle

Hat das Argumentintervall x einen hinreichend großen relativen Durchmesser, so kann wegen Intervallüberschätzungen im Einschließungsalgorithmus der Fall eintreten, daß das Ergebnisintervall `res` einen relativen Durchmesser besitzt, der größer ist als $c2 = 10^{-15}$. Um so große Ergebnisintervalle zu vermeiden, berechnet man eine Einschließung von 10^x, indem man x durch ein Intervall `ti` vom Typ `interval` einschließt und mit `exp10(ti)` eine Einschließung von 10^x im einfachen `real`–Format realisiert. Im Programm erfolgt die entsprechende Abfrage durch die Anweisungen

```
TRANS(res,u1);
kurz := diam(res)/abs(u1) > c2;
if kurz then
begin
   t := told;  goto 1
end;
```

Im Falle `kurz = true` , d.h. bei zu breitem Ergebnisintervall, springt das Programm zum Label 1, wo die oben beschriebenen Intervallrechnungen im einfachen `real`–Format vorgenommen werden. Vergleichen Sie dazu bitte auch den Programmquelltext im nachfolgenden Abschnitt.

Die Prozedur `TRANS(res,u1)` berechnet mit der `real`-Variablen `u1` eine maximalgenaue Näherung des Mittelpunktes von `res`.

1.10.1.9 Numerische Ergebnisse

Mit nachfolgendem Programm lassen sich zu gegebenem staggered Intervallargument x garantierte Einschließungen von 10^x berechnen. Der Anwender kann vorher mit einen `Stagg_prec`–Wert mit: $1 \leq$ `Stagg_prec` ≤ 19 die gewünschte Präzision einstellen, wobei die `integer`–Zahl `Stagg_prec` eine Präzision von $13 \cdot ($`Stagg_prec`$+ 1)$ Dezimalstellen festlegt.

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;

procedure main;
var x,y : Stagg[-1..Stagg_prec];
begin
   repeat
      writeln('x: Stagg  = ?');  read(x);
      y := exp10(x);
      writeln('10^x enthalten in: ');
      writeln(y);
   until false;
end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

Mit den entsprechenden Eingabedaten erhält man die folgenden Ablaufprotokolle:

```
Stagg_prec = ?
3              <enter>
x: Stagg  = ?
 3 Stagg components and 1 interval are needed in this sequence
133.2 0 0 0    <enter>
10^x enthalten in:
 1.584893192461E+133
 1.134852021013E+120
 7.339150701325E+107
[  1.944213382501E+095,  1.944213382507E+095 ]
```

```
Stagg_prec = ?
19     <enter>
x: Stagg  = ?
19 Stagg components and 1 interval are needed in this sequence
133.2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
10^x enthalten in:
 1.584893192461E+133
 1.134852021013E+120
 7.339150701326E+107
 9.442133825039E+094
 6.831629681231E+080
 6.656863668453E+067
 9.801102027238E+054
 4.611104357781E+041
 1.635383559241E+028
 9.865997409063E+015
 4.323960970303E+002
 1.974928168542E-011
 3.481705971899E-024
 3.687619569121E-037
 7.803130286720E-050
 8.004805618887E-063
 4.145367101250E-076
 6.029211383397E-089
 8.690060177890E-102
[  2.628882745237E-114,  2.628882745275E-114 ]

x: Stagg  = ?
6     <enter>
 6 Stagg components and 1 interval are needed in this sequence
-133.2 0 0 0 0 0 [-1e-70,2e-70]   <enter>
10^x enthalten in:
 6.309573444801E-134
 9.324943436013E-147
 6.622343864672E-160
 9.452571882287E-173
 2.452772952883E-186
 3.495015939335E-199
[ -2.282761011000E-203,  2.282761013000E-203 ]
```

1.10.1.10 Der Programm-Quelltext

```
function exp10(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := exp10(x) liefert mit y eine Einschliessung fuer 10^x }
{ Es muss gelten: 1.    x enthalten in [-9.99*10^255,255.4] }
{                 2.                 1 <= Stagg_prec <= +19 }
{ Die erste Bedingung wird nicht ueberprueft!               }
{ Getestete: Ja;                     Blomquist, 15.01.2001; }
label 1,2;
const c1 = 0.17; { Das mit 10^(-k) reduzierte Argument }
         { sollte kleiner sein als 10^(-Stagg_prec*c1) }
   c2 = 1e-15;   { Oberschranke des rel. Durchmessers  }
 { fuer das Argument u. fuer das Ergebnis. Wird dieser }
 { Wert ueberschritten, so erfolgt die Rechnung nur im }
 { einfachen real-Format ab Label 1.                   }
var kurz         : boolean;
   d,u1,u2,u3    : real;
   ti            : interval;
   res,t,PN,told : Stagg[-1..Stagg_prec];
   k,N,j,expo    : integer;
begin
   if Stagg_prec > 19 then s_fu1_math_error(3);
   d := diam(x);    t := x;
   normal(t);  told := t;
   if (d=0) and (SIGN(t[1])=0) then res := 1
      else
   begin
      kurz := false;
      if d > 0 then
      begin
         if 0 in t then kurz := d > c2
         else
            begin
               TRANS(t,u1);
               kurz := d/abs(u1) > c2;
            end;
      end;
1:    if kurz then
      begin { Berechnung von 10^x nur im real-Format: }
```

```
      ti := ival(t);
      res := Staggintval(exp10(ti));
   end   else
   begin { Rel. Durchmesser <= c2 }
      if t[1] < -1e8 then kurz := true
      else
      begin
         expo := round(t[1]); { expo: Gerundeter ganzzahl. }
         kurz := expo < -267; {       Exponent.            }
      end;
      if kurz then goto 1;
      t := t - expo;
      if (t[1] = 0) and (diam(t)=0) then
      begin PN := 1; goto 2 end;
      t := t * change_length(ln10);
      res := sup( abs(t) ) + 0;
      k := trunc(c1*Stagg_prec + log10(res[1]));
      if k < 0 then k := 0;
      if k = 0 then k := 1;
      t := t M_10HOCH -k; { t: Mit 10^(-k) reduz. Argument }
      { Berechnung des Polynomgrades N:     }
      u1 := 1 - 13*(Stagg_prec+1);
      u3 := sup( abs(ival(t)) );  u2 := log10(u3);
      N := 1;  d := 1;
      repeat
         d := d*(N+1)*(N+2)*(N+3);
         N := N + 3;
      until u1 + log10(d*(N+1)) > (N+1)*u2;
      { Garantierte Oberschr. des abs. Approxim.-Fehlers: }
      ti := u3;                           { ti : interval }
      ti := power(ti,N+1) / (factorial_h(N+1)*(1-ti));
      { Auswertung des Approximationspolynoms ohne die 1: }
      PN := t / N;
      for j := N-1 downto 1 do PN := ( PN+1 )*t / j;
      PN := PN + 1 + Staggintval(-ti.sup,ti.sup);
      { Jetzt wurde neben der 1 auch der absolute }
      { Approximationsfehler beruecksichtigt!     }
      { Jetzt folgt die Ruecktransformation der   }
      { Argumentreduktion mit dem Faktor 10^(-k): }
      k := trunc( comp(0.1,k+1) );
```

```
         PN := power(PN,k);
2:       res := PN M_10HOCH expo;    { Multiplik. mit 10^expo }
         { Falls Intervallaufblaehung zu gross, dann Rechnung }
         { nur im einfachen real-Format:                      }
         TRANS(res,u1);
         kurz := diam(res)/abs(u1) > c2;
         if kurz then
         begin
            t := told;  goto 1
         end;
      end;
   end;
   exp10 := res;
end; { exp10 }
```

1.10.2 e^x

Zwischen den Exponentialfunktionen zur Basis e und 10 besteht folgender Zusammenhang:

$$e^x = 10^{x \cdot \log_{10}(e)} \tag{1.68}$$

Ist `exp10(x)` eine Einschließung von $10^x \subseteq$ `exp10(x)` und ist die staggered Variable `log10e` eine Einschließung von $\log_{10}(e) \subseteq$ `log10e`, so gilt nach obiger Gleichung (1.68)

$$e^x \subseteq \mathtt{exp10}(\mathtt{x} * \mathtt{log10e}) \tag{1.69}$$

Zu gegebenem staggered Argument x liefert die folgende Funktion

```
function exp(var x : Stagg) : Stagg[-1..Stagg_prec];
begin
   exp := exp10(x*log10e);
end; { exp }
```

eine garantierte Einschließung von $e^x \subseteq$ `exp(x)`. Das nachfolgende Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;

procedure main;
var x,y : Stagg[-1..Stagg_prec];
begin
   repeat
      writeln('x: Stagg  = ?');  read(x);
      y := exp(x);
      writeln('e^x enthalten in: ');
      writeln(y);
   until false;
end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefer mit den entsprechenden Eingabedaten das folgende Ablaufprotokoll:

```
Stagg_prec = ?
19     <enter>
x: Stagg  = ?
19 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
e^x enthalten in:
 2.718281828459E+000
 4.523536028747E-014
 1.352662497757E-027
 2.470936999595E-040
 7.496696762772E-053
 4.076630353547E-066
 5.945713821785E-079
 2.516642742746E-092
 6.391932003059E-105
 9.218174135966E-118
 2.904357290033E-131
 4.295260595630E-144
 7.381323286279E-157
 4.349076323382E-170
 9.880753195251E-183
 1.901157383418E-197
 7.930702154089E-210
 1.499348841675E-223
 9.244761460665E-237
[  3.082264799848E-249,  3.082264800591E-249 ]
```

Damit haben wir die Zahl $e = e^1$ mit einer Genauigkeit von fast $13\cdot(19+1) = 260$ Dezimalziffern eingeschlossen. In allen praktischen Anwendungen wird diese maximale Genauigkeit mit `Stagg_prec=19` völlig ausreichend sein.

1.10.3 2^x

Zwischen den Exponentialfunktionen zur Basis 2 und 10 besteht folgender Zusammenhang:

$$2^x = 10^{x \cdot \log_{10}(2)} \tag{1.70}$$

Ist `exp10(x)` eine Einschließung von $10^x \subseteq$ `exp10(x)` und ist die staggered Variable `log102` eine Einschließung von $\log_{10}(2) \subseteq$ `log102`, so gilt nach obiger Gleichung (1.70)

$$2^x \subseteq \mathtt{exp10}(\mathtt{x} * \mathtt{log102}) \tag{1.71}$$

Zu gegebenem staggered Argument x liefert die folgende Funktion

```
function exp2(var x : Stagg) : Stagg[-1..Stagg_prec];
begin
   exp2 := exp10(x*log102);
end; { exp2 }
```

eine garantierte Einschließung von $2^x \subseteq$ `exp2(x)`. Das folgende Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;

procedure main;
var x,y : Stagg[-1..Stagg_prec];
begin
   repeat
      writeln('x: Stagg  = ?');  read(x);
      y := exp2(x);
      writeln('2^x enthalten in: ');
      writeln(y);
   until false;
end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefer mit den entsprechenden Eingabedaten das folgende Ablaufprotokoll:

```
Stagg_prec = ?
19     <enter>
x: Stagg  = ?
 19 Stagg components and 1 interval are needed in this sequence
1.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
2^x enthalten in:
 2.828427124746E+000
 1.900976033774E-013
 4.841939615713E-026
 9.343750753896E-039
 1.463533594759E-052
 8.146495692421E-065
 4.077700775068E-078
 6.552831454700E-091
 2.769246182459E-104
 4.049849672111E-117
 7.014744252882E-130
 4.299419987166E-143
 2.826445331855E-156
 1.118551159990E-170
 1.002305564121E-183
 1.429402191199E-196
 4.321194054906E-209
 9.193724029457E-222
 3.483728177839E-236
[  7.219104658259E-249,  7.219104658620E-249 ]
```

Damit haben wir die Zahl $2^{3/2} = \sqrt{8}$ mit einer Genauigkeit von nahezu $13 \cdot (19+1) = 260$ Dezimalziffern eingeschlossen. In allen Anwendungen wird diese maximale Genauigkeit mit `Stagg_prec=19` völlig ausreichend sein.

1.10.4 $e^x - 1$

Ist der Funktionswert $e^x - 1$ für $|x| \ll 1$ zu berechnen, so erhält man umso ungenauere Ergebnisse, je mehr sich x der Zahl Null nähert. In diesem Fall benutzt man zur Approximation von $e^x - 1$ ein abgebrochene Taylorreihe mit einem umso größerern Polynomgrad N, je größer die geforderte Genauigkeit ist. Mit ε_0 bezeichnen wir die Oberschranke des Betrages des absoluten Approximationsfehlers. Bedeutet x das staggered Intervallargument und ist y die staggered Einschließung von $e^x - 1$ auf der Maschine, so lautet der Grobalgorithmus:

$$\texttt{if } x \subset [-10^{-13}, +10^{-13}] \texttt{ then } e^x - 1 \subset y = \sum_{n=1}^{N} \frac{x^n}{n!} + [-\varepsilon_0, +\varepsilon_0]$$

$$\texttt{else}$$

$$\texttt{if } x \subset (-\infty, -1) \texttt{ or } x \subset (1, +\infty) \texttt{ then } y := \texttt{exp(x)}-1$$

$$\texttt{else } y := \texttt{exp(x)}-1;\ \text{Auswertung in höherer Präzision.}$$

Der Funktionsaufruf `exp(x)` liefert dabei auf der Maschine die garantierte Einschließung von $e^x \subseteq$ `exp(x)`. Die Auswertung von `exp(x)` $- 1$ in den Teilbereichen $[-1, -10^{-13}]$ und $[10^{-13}, +1]$ ist in höherer Präzision notwendig `(set_Stagg_prec(p+1))`, um die durch `Stagg_prec` geforderte Genauigkeit zu erfüllen. Im Programm wird das Intervall $[-\varepsilon_0, +\varepsilon_0]$ realisiert durch den Funktionsaufruf: `Staggintval(-ti.sup,ti.sup)`, vergl. den Quelltext ab Seite 96.

1.10.4.1 Approximation von $e^x - 1$

Im Fall $x \subset [-10^{-13}, +10^{-13}]$ wird $e^x - 1$ approximiert durch das Taylorpolynom $P_N(x)$ vom Grade N:

$$e^x - 1 \approx P_N(x) := \sum_{n=1}^{N} \frac{x^n}{n!}, \qquad N = 1, 2, \ldots$$

Ist nun ζ_x eine Oberschranke der Beträge aller Elemente aus x, so gilt für den absoluten Approximationsfehler, wie bei der Exponentialfunktion von Seite 80, für alle $\zeta \in x$ die Abschätzung:

$$\left| (e^{\zeta} - 1) - \sum_{n=1}^{N} \frac{\zeta^n}{n!} \right| \leq \frac{\zeta_x^{N+1}}{(N+1)!} \cdot \frac{1}{1 - \zeta_x}, \quad \zeta_x := \sup(\text{abs}(x)); \tag{1.72}$$

Um eine garantierte Oberschranke des absoluten Approximationsfehlers zu erhalten, muß die rechte Seite von (1.72) intervallmäßig ausgewertet werden.

Aus Laufzeitgründen wird man diese Auswertung jedoch nicht im staggered Format sondern im einfachen `interval`–Format vornehmen. Im Programm wird ζ_x durch die `real`–Variable `zx` realisiert.

1.10.4.2 Der Polynomgrad N

Zu gegebenem `Stagg_prec`–Parameter S und zu gegebener Oberschranke $\zeta_x < 10^{-13}$ ist jetzt der Polynomgrad N so zu bestimmen, daß möglichst die letzte Dezimalziffer der Einschließung von $e^x - 1$ korrekt berechnet wird.

Im folgenden Beispiel betrachten wir $\zeta_x = 10^{-15}$, dann liegt $e^x - 1$ in der gleichen Größenordnung, und mit $S = 4$ müßten dann $D = 13 \cdot (4+1)$ Dezimalziffern berechnet werden. Für den absoluten Approximationsfehler ε_{abs} bedeutet dies: $|\varepsilon_{abs}| < 10^{-15-D}$ oder allgemein:

$$(1.73) \qquad |\varepsilon_{abs}| < 10^{\mathtt{expo}(\zeta_x)-1-D}, \quad D = 13 \cdot (S+1);$$

Da im `real`–Format der Betrag des absoluten Fehlers ε_{abs} den Wert 10^{-267} nicht unterschreiten kann, muß der Exponent in (1.73) größer sein als 10^{-267}. Mit der `integer`–Größe

$$k := \max\{\mathtt{expo}(\zeta_x) - 1 - D, -267\} \quad \text{und} \quad D = 13 \cdot (S+1)$$

ergibt sich dann für den Betrag des absoluten Approximationsfehlers mit den Ungleichungen (1.72), (1.73) die vereinfachte[5] Bedingung:

$$(1.74) \qquad |\varepsilon_{abs}| \leq \frac{\zeta_x^{N+1}}{(N+1)!} < 10^k, \quad k := \max\{\mathtt{expo}(\zeta_x) - 1 - D, -267\}$$

In logarithmischer Form lautet die Forderung:

$$(1.75) \qquad k + \log_{10}(N+1)! > (N+1) \cdot \log_{10}(\zeta_x)$$

Beginnend mit $N = 1$ wird N in einer `repeat`–Schleife im einfachen `real`–Format schrittweise um 3 erhöht, bis (1.75) erfüllt ist; dabei werden k und $\log_{10}(\zeta_x)$ in den `real`–Variablen `k` und `v` gespeichert. Beachten Sie bitte, daß (1.75) aus Laufzeitgründen im einfachen `real`–Format ausgewertet wird. Mit dem berechneten Polynomgrad N läßt sich dann nach intervallmäßiger Auswertung von (1.72) eine garantierte Oberschranke des absoluten Approximationsfehlers bestimmen. Im Programm wird diese Oberschranke in `ti.sup` fesgehalten, so das der absolute Approximationsfehler im staggered Format durch das Intervall `Staggintval(-ti.sup,ti.sup)` eingeschlossen wird.

[5] Der Faktor $1/(1-\zeta_x)$ in (1.72) kann in guter Näherung gleich 1 gesetzt werden!

1.10.4.3 Auswertung des Taylorpolynoms

Nach der Bestimmung des Polynomgrades N muß das Taylorpolynom $P_N(x)$ für alle alle $\zeta \in x$ eingeschlossen werden. Es gilt:

$$P_N(x) := \sum_{n=1}^{N} \frac{x^n}{n!} = \left(\dots \left(\left(\frac{x}{N} + 1 \right) \frac{x}{N-1} + 1 \right) \dots + 1 \right) \frac{x}{2}$$

Wertet man das Schema oben rechts in der staggered Intervall–Arithmetik aus und speichert das Ergebnis in der Variablen `res`, so gilt: $P_N(x) \subseteq \texttt{res}$. Addiert man zusätzlich noch das Intervall `Staggintval(-ti.sup,ti.sup)`, das den absoluten Approximationsfehler enthält, so gilt:

$$e^x - 1 \subseteq \texttt{res} + \texttt{Staggintval}(-\texttt{ti.sup}, \texttt{ti.sup}),$$

womit dann eine garantierte Einschließung von $e^x - 1$ berechnet wird.

1.10.4.4 Numerische Ergebnisse

Mit $x = 10^{-15}$ und `Stagg_prec` $= 16$ erhält man für $e^x - 1$ die garantierte und maximalgenaue Einschließung:

```
    e^x - 1  enthalten in:
     1.000000000000E-015
     5.000000000000E-031
     1.666666666666E-046
     6.670833333333E-059
     3.333341666666E-072
     6.666666680555E-085
     5.555555555575E-098
     3.968253968253E-111
     9.930555555555E-124
     5.555831128747E-137
     7.954144896384E-150
     4.797178130761E-163
     9.849286515953E-176
     3.913874191651E-189
     9.695903376458E-202
     9.320146022770E-215
 [   4.991028959359E-227,  4.991028959360E-227 ]
```

1.10.4.5 Der Programm-Quelltext

```
function expm1(var x : Stagg) : Stagg[-1..Stagg_prec];
const c1 = 1e-13;
      c2 = 1;
var t,res   : Stagg[-1..Stagg_prec];
   t1       : Stagg;
   u,v,d    : real;
   p,ex,k,N : integer;
   ti       : interval;
begin
   if Stagg_prec > 18 then s_fu1_math_error(4);
   t := x;    { x darf sich nicht aendern!                 }
   normal(t); { Die t[j] ueberlappen sich jetzt nicht mehr! }
   u := t[1] + t[-1];  v := t[1] + t[0];
   if (u > -c1) and (v < c1) then
   begin { Reihenentwicklung }
      if (t[1] = 0) and (diam(t)=0) then res := 0
      else
      begin    { t <> Null-Intervall, Reihenentwicklung: }
         u := sup( abs(ival(t)) );
         { Approx-Fehler spaeter mit diesem u berechnen! }
         k := expo(u) - 1 - 13 * (Stagg_prec + 1);
         if k < -267 then k := -267;
         v := log10(u);
         N := 1;  d := 1;
         repeat
            d := d*(N+1)*(N+2)*(N+3);
            N := N + 3;
         until k + log10(d*(N+1)) > (N+1)*v;
         { Absoluter Approximationsfehler: }
         ti := u;
         ti := power(ti,N+1) / (factorial_h(N+1)*(1-ti));
         { Polynomauswertung: }
         res := t / N;
         for p := N-1 downto 1 do res := ( res+1 )*t / p;
         { Absoluten Approximationsfehler beruecksichtigen: }
         res := res + Staggintval(-ti.sup,ti.sup);
      end;
   end                 else
```

```
   begin { e^t - 1 = e^t - 1 }
      if (v < -c2)  or (u > c2) then res := exp(t)-1
      else
      begin        { Berechnung von e^t - 1 mit der }
         { Praezision: Stagg_prec := Stagg_prec + 1 }
         p := get_Stagg_prec;
         allocate(t1,-1..p+1);
         set_Stagg_prec(p+1);
         t1 := change_length(t);
         t1 := exp(t1) - 1;
         set_Stagg_prec(p);  { wieder alte Praezision: }
         res := change_length(t1);
         free(t);
      end;
   end;
   expm1 := res;
end; { expm1 }
```

1.10.5 e^{-x^2}

1.10.5.1 Der Algorithmus

Mit dem staggered Intervallargument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ gilt zunächst die Definition:

$$e^{-x^2} := \left\{ t \,|\, t = e^{-\zeta^2};\ \zeta \in x \subseteq (-\infty, +\infty) \right\}$$

Da $e^{-\zeta^2}$ eine gerade Funktion ist, kann man sich beim Auftreten negativer ζ-Werte beschränken auf das Intervallargument $|x|$:

$$e^{-x^2} := \left\{ t \,|\, t = e^{-\zeta^2};\ \zeta \in x \subseteq [0, +\infty) \right\}$$

Für $\zeta > 25$ liefert die Auswertung von $e^{-\zeta^2}$ einen Underflow, so daß bei der Intervallauswertung die Untergrenze des Ergebnisintervalls e^{-x^2} auf Null gesetzt werden muß. Im `real`–Format besitzt das Ergebnisintervall daher die Mindestbreite 10^{-267} und kann daher auch für Punktargumente x nicht mehr beliebig genau berechnet werden. Aus diesem Grunde erfolgt daher im Fall $\zeta > 25$ die Einschließung von e^{-x^2} auf der Maschine durch Auswertung der schon vorhandenen Intervallfunktion `exp_x2` im einfachen `real`–Format. Sonst wird e^{-x^2} eingeschlossen durch den Funktionsaufruf `exp(-x*x)`; falls dabei der maximale Funktionswert 1 überschritten wird, setzt man die Oberschranke des Ergebnisintervalls auf 1.

1.10.5.2 Der Programm-Quelltext

```
function exp_x2(var x : Stagg) : Stagg[-1..Stagg_prec];
{ y := exp_x2(x)  --->  y ist Einschliessung fuer  e^(-x^2) }
var t : Stagg[-1..Stagg_prec];
begin
   t := x;  normal(t);
   if SIGN(t[1]+t[-1]) < 0 then t := abs(t);
   if t[1]+t[0] > 25 then t := Staggintval( exp_x2(ival(t)) )
   else  begin
            t := exp(-t*t);
            if sup(t) > 1 then { e^(-r^2) <= 1 !! }
               t := hull( inf(t),Staggintval(1) );
         end;    exp_x2 := t;
end; { exp_x2 }
```

1.10.5.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
{ Einschliessung von exp(-x^2) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       y := exp_x2(x);
       writeln('e^(-x^2)  enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man für die Einschließung von e^{-x^2} die folgenden Ergebnisse:

```
Stagg_prec = ?
7      <enter>
x: Stagg   = ?
7 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0 0 [0,1e-267]     <enter>
e^(-x^2)  enthalten in:
 9.999999999999E-001
 9.999999999999E-014
 9.999999999999E-027
 9.999999999999E-040
 9.999999999999E-053
 9.999999999999E-066
 9.999999999999E-079
[  9.999999999999E-092,  1.000000000000E-091 ]
```

```
Stagg_prec = ?
19      <enter>
x: Stagg   = ?
19 Stagg components and 1 interval are needed in this sequence
0.1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
e^(-x^2)  enthalten in:
 9.900498337491E-001
 6.805357390597E-014
 7.180036557772E-027
 7.908125383746E-041
 6.883878745293E-054
 1.477271687452E-067
 9.501821553077E-080
 9.383810992963E-093
 3.337294097244E-106
 8.388834834583E-119
 6.489260894054E-132
 8.417118727078E-145
 2.585794472837E-158
 2.134431402601E-172
 9.901117654843E-185
 8.880124939407E-198
 2.329456107103E-212
 1.440822488235E-225
 3.623410466435E-238
[  4.103730928282E-251,  4.103730928781E-251 ]


Stagg_prec = ?
3       <enter>
x: Stagg   = ?
3 Stagg components and 1 interval are needed in this sequence
25.5 0 0 0      <enter>
e^(-x^2)  enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  1.000000000000E-267 ]
```

1.10.6 y^x

Zu gegebenen staggered Argumenten $x = x_1 + x_2 + \ldots + x_{\text{ub}(x)} + [x_{-1}, x_0]$ und $y = y_1 + y_2 + \ldots + y_{\text{ub}(y)} + [y_{-1}, y_0]$ ist einzuschließen der Wertebereich

$$(1.76) \quad x^y := \begin{cases} 1\,, & \text{falls } y = 0 \text{ und } 0 \notin x,\ (k = 1) \\ x\,, & \text{falls } y = 1,\ (k = 2) \\ 0\,, & \text{falls } x = 0 \text{ und } \inf(y) > 0,\ (k = 3) \\ x^n\,, & \text{falls } y = n \in \mathbb{Z} \ \text{ und } |n| < 10^9,\ (k = 4) \\ e^{y \cdot \ln(x)}\,, & \text{falls } \inf(x) > 0,\ (k = 5) \\ \text{Fehlermeldung} & \text{sonst; } (k = 0) \end{cases}$$

Im Programm werden obige Bedingungen, die durch $k = 1, 2, 3, 4, 5$ gekennzeichnet sind, genau in dieser Reihenfolge abgefragt. Man erhält also mit der existierenden Potenz $(-1)^{1000000000} = 1$ eine Fehlermeldung, da $x = [-1, -1]$ und $y = [10^9, 10^9]$ durch keine der fünf Bedingungen $k = 1, 2, 3, 4, 5$ abgedeckt wird und somit eine Fehlermeldung erzeugt. Potenzen mit einer Basis, die auch negative Elemente enthalten, werden also für ganzzahlige Exponenten n nur dann berechnet, wenn die Bedingung $|n| < 10^{+9}$ erfüllt ist. Eine wirkliche Einschränkung für die numerische Praxis bedeutet dies natürlich nicht! Zur Vermeidung stärkerer Intervallüberschätzungen wird in den Fällen $k = 4, 5$ in erhöhter Präzision gerechnet, so daß die Einschließungen von x^y meist maximalgenau erfolgen. Weitere Einzelheiten findet man im folgenden Quelltext der Funktion `power(x,y)` $\supseteq x^y$.

```
function power(var x,y : Stagg) : Stagg[-1..Stagg_prec];
{ power(x,y) liefert Einschliessung von x^y }
var k     : integer;
   t      : Stagg[-1..Stagg_prec];
   x1,y1  : Stagg;
   ok,bl  : boolean;
begin
   if Stagg_prec > 17 then s_fu1_math_error(11);
   k := 0;
   if (y=0) and not (0 in x) then k := 1 else
      if y=1 then k := 2 else
         if (x=0) and (y>0) then k := 3;
   if k=0 then
   begin
      t := y;  Normal(t);
```

```
      ok := (t[-1]=t[0]) and (t[0]=0);
      if Stagg_prec > 1 then ok := ok and (t[2]=0);
      ok := ok and (abs(t[1])<1e9);
      if ok then ok := Entire(t[1],bl);
      if ok then k := 4 else
         if x > 0 then k := 5;
   end;
   case k of
     1 : t := 1;
     2 : t := x;
     3 : t := 0;
     4 : begin
            k := Stagg_prec;
            allocate(x1,-1..k+1);  allocate(y1,-1..k+1);
            set_Stagg_prec(k+1);
            x1 := change_length(x);
            y1 := power(x1,trunc(t[1]));
            set_Stagg_prec(k);
            t := change_length(y1);  free(x1);  free(y1);
         end;
     5 : begin
            k := Stagg_prec;
            allocate(x1,-1..k+1);  allocate(y1,-1..k+1);
            set_Stagg_prec(k+1);
            x1 := change_length(x);  Normal(x1);
            y1 := change_length(y);
            if x1[1]=1 then
            begin
               x1[1] := 0;  x1 := ln1p(x1);
            end            else x1 := ln(x1);
            x1 := exp(y1 * x1);
            set_Stagg_prec(k);
            t := change_length(x1);  free(x1);  free(y1);
         end;
     0 : s_fu1_math_error(10);
   end; { case }
   power := t;
end; { power }
```

Numerische Ergebnisse:

Das **PASCAL-XSC** Programm

```
program tests;
{ y := power(x,y) liefert Einschliessung von x^y }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var t,x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg   = ?');   read(x);
       writeln('y: Stagg   = ?');   read(y);
       t := power(x,y);
       writeln('x^y  enthalten in: ');
       write(t);
       writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für die Potenz x^y die folgenden Einschließungen:

```
Stagg_prec = ?
3      <enter>
x: Stagg   = ?
3 Stagg components and 1 interval are needed in this sequence
1 0 0 1e-150    <enter>
y: Stagg   = ?
3 Stagg components and 1 interval are needed in this sequence
1e9 0 0 1e-267  <enter>
x^y  enthalten in:
 1.000000000000E+000
 9.999999999999E-142
 9.999999999999E-155
[  9.999999999999E-168,  1.000000000001E-167 ]
```

```
Stagg_prec = ?
17     <enter>
x: Stagg   = ?
17 Stagg components and 1 interval are needed in this sequence
22.34 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
y: Stagg   = ?
17 Stagg components and 1 interval are needed in this sequence
-33.221 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
x^y  enthalten in:
 1.520925880290E-045
 1.680931390307E-058
 3.847850465596E-071
 3.988343882281E-084
 7.703998505437E-097
 8.591575206342E-110
 4.561358820292E-124
 5.856298472674E-138
 7.168457326690E-151
 4.123441516823E-164
 4.535558153641E-177
 3.069480735944E-190
 9.996094263775E-203
 8.865699737622E-216
 8.904508781898E-229
 3.993764897613E-242
 0.000000000000E+000
[  9.089768130151E-255,  9.089768130154E-255 ]

Stagg_prec = ?
2      <enter>
x: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-2,1]     <enter>
y: Stagg   = ?
2 Stagg components and 1 interval are needed in this sequence
400 0 0        <enter>
x^y  enthalten in:
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  2.582249878090E+120 ]
```

Das folgende **PASCAL-XSC** Programm

```
program tests;
{ Einschliessung von e^(sqrt[7](Pi)) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var t : Stagg[-1..Stagg_prec];
 begin
    t := power(exp(Staggintval(1)),
                   sqrtn(change_length(Pi_sm),7));
    writeln('e^(sqrt[7](Pi)) enthalten in: ');
    write(t);
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

berechnet eine garantierte Einschließung von $e^{\sqrt[7]{\pi}}$

```
Stagg_prec = ?
12
e^(sqrt[7](Pi)) enthalten in:
 3.246780954711E+000
 2.000620590106E-013
 9.763608983188E-026
 4.321388461610E-039
 5.325007930549E-052
 8.594815589863E-065
 7.291172035602E-078
 5.344667671885E-091
 9.690215635915E-104
 6.250722537575E-117
 8.426783343274E-130
 5.061947861118E-143
[  8.271560926217E-156,  8.271560926431E-156 ]
```

1.10.7 Die trigonometrischen Funktionen

1.10.7.1 sin(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen sin–Funktion:

$$\sin(x) := \{y \,|\, y = \sin(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung ub(x) = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\sin(x)$ erfolgt mit einer Präzision von mindestens $13\cdot(S{+}1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.7.1.1 Der Algorithmus

Zunächst wird der Wertebereich $\sin(x)$ mit Hilfe der schon implemntierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\sin(-x) \equiv -\sin(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\sin(x)$ eine garantierte Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Argumentreduktion:

Für alle ganzen Zahlen $m \in \mathbb{Z}$ gilt mit $y := x - m \cdot \pi$ zunächst die Beziehung:

$$\sin(x) = \begin{cases} \sin(y) & \text{falls } m \text{ gerade} \\ -\sin(y) & \text{falls } m \text{ ungerade} \end{cases}, \qquad y := x - m \cdot \pi;$$

Berechnet man m jedoch speziell durch: `m := round(x[1]/π)`, so gilt für das reduzierte Argument y

$$y := x - m \cdot \pi \Subset [-\frac{\pi}{2}, +\frac{\pi}{2}],$$

wobei das Symbol $\Subset$ bedeutet, daß $\inf(y)$ wegen der endlichen Breite des Intervalls y den Wert $-\pi/2$ leicht nach unten oder $\sup(y)$ den Wert $+\pi/2$

auch leicht nach oben überschreiten kann. Im Programm wird y in maximaler Präzision $S :=$ `Stagg_prec` $= 20$ berechnet, um Auslöschungseffekte und die damit verbundenen Intervallüberschätzungen möglichst zu vermeiden. Bitte beachten Sie in diesem Zusammenhang, daß man π im `real`–Format mit einer Präzision $S > 20$ nicht genauer einschließen kann als mit dem Wert `Pi_sm`, der mit der Präzision $S = 20$ im Modul `stagg` realisiert wurde.

Im Falle $0 \in y$ wird $\sin(y)$ wieder mit der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen und wegen $\sin(-y) \equiv -\sin(y)$ besteht die Aufgabe jetzt darin, eine garantierte Einschließung für den Wertebereich $\sin(y)$ zu berechnen unter der Voraussetzung $y \Subset (0, \pi/2]$.

Nach dieser ersten Argumentreduktion kann $\sup(y)$ aber immer noch etwas größer ausfallen als $\pi/2$, so daß in diesem Fall eine Approximation von $\sin(y)$ mit Hilfe einer abgebrochenen Taylor–Reihe viel zu aufwendig wäre. Bei einer zweiten Argumentreduktion benutzen wir die Funktionalgleichung

$$\sin(y) = \sin\left(\frac{y}{3}\right) \cdot \left[3 - 4 \cdot \sin^2\left(\frac{y}{3}\right)\right] \tag{1.77}$$

Ersetzt man in (1.77) das Intervall y durch $y/3$, so erhält man für $\sin(y/3)$ einen Funktionsausdruck in $\sin(y/3^2)$. Ersetzt man dann $\sin(y/3)$ rechts in (1.77) durch diesen Funktionsterm, so erhält man für $\sin(y)$ einen Ausdruck in $\sin(y/3^2)$. Nach p Schritten erhält man dann für $\sin(y)$ einen Term in $\sin(y/3^p)$, und mit hinreichend großem p liegt das reduzierte Intervall-Argument $y/3^p$ beliebig dicht am Ursprung. Im Programm wählen wir p so groß, daß $y/3^p \subset (0, 10^{-4}]$ erfüllt wird.

Ist dann t eine Einschließung von $\sin(y/3^p) \subseteq t$, die durch Approximation durch ein Taylorpolynom und durch Berücksichtigung des Approximationsfehlers berechnet wurde, so erhält man für $\sin(y)$ eine Einschließung durch folgende Schleife:

```
For j := 1 to p do t := t*(3-4*(t*t));
```
$\rightsquigarrow \quad \sin(y) \subseteq$ `t`

Abschätzung des Approximationsfehlers:

$$\text{Für } r \in \mathbb{R} \text{ gilt:} \qquad \sin(r) = r \cdot \sum_{k=0}^{\infty} (-1)^k \cdot \frac{r^{2k}}{(2k+1)!}$$

Mit dem Approximationspolynom vom Grade N

$$P_N(r) := r \cdot \sum_{k=0}^{N} (-1)^k \cdot \frac{r^{2k}}{(2k+1)!}$$

erhält man dann für den Betrag des absoluten Approximationsfehlers ε_a für $r > 0$ die Abschätzung:

$$\begin{aligned} |\varepsilon_a| &= |\sin(r) - P_N(r)| = r \cdot \left| \sum_{k=N+1}^{\infty} (-1)^k \cdot \frac{r^{2k}}{(2k+1)!} \right| \\ &\leq r \cdot \frac{r^{2N+2}}{(2N+3)!} = \frac{r^{2N+3}}{(2N+3)!}, \end{aligned}$$

dabei ergibt sich die letzte Abschätzung aus der Tatsache, daß die Potenz-Reihe der sin-Funktion eine Leibniz-Reihe ist.

Bedeutet dann $t := y/3^p$ das Argument–Intervall, in dem $\sin(t)$ durch den Wertebereich $P_N(t)$ zu approximieren ist, so gilt:

$$\sin(t) \subseteq P_N(t) + [-\texttt{eps}, +\texttt{eps}], \qquad \texttt{eps} := \frac{t.\texttt{sup}^{2N+3}}{(2N+3)!} \tag{1.78}$$

Berechnung des Polynomgrades N:

Die Approximation von $\sin(y/3^p)$ erfolgt für $y/3^p \subset (0, 10^{-4}]$, so daß mit $\zeta \in y/3^p$ gilt[6]:

$$\sin(\zeta) \approx \zeta$$

Zu gegebener Präzision S := `Stagg_prec` muß im `real`–Format für den Betrag des absoluten Fehlers Δ gelten:

$$|\Delta| \approx 10^p, \quad \text{mit} \quad p := \text{expo}(\zeta) - 1 - 13 \cdot (S+1);$$

Da im `real`–Format 10^{-267} die kleinste darstellbare positive Zahl ist, macht es keinen Sinn, p kleiner zu wählen als -267. Im Falle

$$\text{expo}(\zeta) - 1 - 13 \cdot (S+1) < -267$$

wird daher $p = -267$ gesetzt. Der Polynomgrad N ergibt sich dann aus der Forderung:

$$\frac{\zeta^{2N+3}}{(2N+3)!} < 10^p \qquad \Longleftrightarrow$$

$$(2N+3) \cdot \log_{10}(\zeta) - \log_{10}(2N+3)! < p \tag{1.79}$$

Im Programm wird N schrittweise um 1 erhöht, bis (1.79) erfüllt ist.

[6] Im Programm ist $\zeta = \sup(y/3^p)$

Polynomauswertung:
Mit dem Intervallargument $x = y/3^p$ gilt:

$$P_N(x) = x \cdot \sum_{k=0}^{\infty} (-1)^k \cdot \frac{x^{2k}}{(2k+1)!}, \quad \text{und} \quad \sum_{k=0}^{\infty} (-1)^k \cdot \frac{x^{2k}}{(2k+1)!} =$$
$$= \left(\cdots \left(\left(\frac{-x^2}{(2N+1)\cdot 2N} + 1 \right) \cdot \frac{-x^2}{(2N-1)(2N-2)} + 1 \right) \cdots + 1 \right) \cdot \frac{-x^2}{3\cdot 2} + 1$$

Auf der Maschine wird die Auswertung der letzten Summe daher realisiert durch die einfache Schleife:

```
y := -x*x;  P := 1;
For j := N downto 1 do
begin
   k := 2*j;  pot := k*(k+1);
   P := P * y / pot + 1;
end;
P := x * P;   { P: Approximation fuer sin(x) }
```

Nach (1.78) liefert dann die Anweisung

```
P := P + Staggintval(-eps,eps);
```

mit `P` eine garantierte Einschließing von $\sin(x) \subseteq$ `P`.

1.10.7.1.2 Der Programm-Quelltext

```
procedure sin_N_e(    S : integer; u : real;
                  var N : integer; var e : real);
{ Berechnet Polynomgrad N und Oberschranke }
{ e des absoluten Approximationsfehlers.   }
{ u muss x.sup sein!!!                     }
var lgu,k : real;
   p      : integer;
   xi     : interval;
begin
   lgu := log10(u);
   p := expo(u) - 1 - 13*(S+1);
   if p < -267 then p := -267;
   N := 0;  k := 3;
   repeat
```

```
      N := N+1;
      k := k *< (2*N+2)*(2*N+3);
   until (2*N+3)*lgu - log10(k) < p;
   xi := intval(u);
   xi := power(xi,2*N+3) / intval(k);
   e := xi.sup;
end;

function sin_ge0(var x : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet Einschliessung von sin(x), wobei x ein }
{ schmales Intervall ist, mit:   inf(x) > 0;       }
{ Die x[j] ueberlappen sich nicht fuer j=1,2,...   }
const c = 1e-4;
   rlg3 = 2.096; { rlg3 > 1/log_10(3) }
var N,S,j,pot,p,k   : integer;
   y,t,x1           : Stagg[-1..Stagg_prec];
   y1               : Stagg[-1..20];
   xi               : interval;
   eps              : real;
   negm,negy,reduce,testp : boolean;
begin
   if x[1] > 6.746e9 then s_fu1_math_error(14);
   N := round(x[1]/PI_1);
   negm := N mod 2 <> 0;  negy := false;  testp := false;
   { 1. Argumentreduktion in maximaler Praezision: }
   S := get_Stagg_prec;
   set_Stagg_prec(20);      { Maximale Praezision! }
   y1 := change_length(x);
   y1 := y1 - N * Pi_sm;
   reduce := abs(y1[1]) > c;
   { reduce = true  <===>  2. Arg.-Reduktion erforderlich }
   if (Sign(y1[-1]) <> Sign(y1[0])) and (Sign(y1[1])=0)
      then { 0 in y1 enthalten! } t := sin( ival(y1) ) else
      begin
         negy := SIGN(y1[1]+y1[-1]) < 0;
         if negy then y1 := -y1; { Ab jetzt: inf(y1) > 0 }
         set_Stagg_prec(S);
         x1 := change_length(y1);
         testp := x1[1] > 1.570796326794;
         if reduce then
```

```
         begin { 2. Arg.-Reduktion ist notwendig! }
            eps := rlg3 * (4 + log10(x1[1]));
            p := trunc(eps) + 1;
            j := round( power(3,p) );
            x1 := x1 / j;  { 2. Argument-Reduktion }
         end;
         { Berechnung des Polynomgrades: }
         xi := ival(x1);
         sin_N_e(S,xi.sup,N,eps);
         { Polynomauswertung: }
         y := -x1*x1;  t := 1;
         For j := N downto 1 do
         begin
            k := 2*j;  pot := k*(k+1);
            t := t*y/pot + 1;
         end;
         t := x1 * t;        { t = Approximation fuer sin(x1) }
         t := t + Staggintval(-eps,eps);     { Approx-Fehler! }
         if reduce then   { Ruecktransformation: }
            For j := 1 to p do t := t*(3-4*(t*t));
      end;
   set_Stagg_prec(S);
   if negm then
   begin
      if not negy then t := -t
   end     else if negy then t := -t;
   if testp then { Test auf:   -1 <= sin(x) <= +1 }
   begin
      y := Sup(t);
      if y > 1 then t := hull(Inf(t),Staggintval(1))
      else
         if Inf(t) < -1 then t := hull(Staggintval(-1),y);
   end;
   sin_ge0 := t;
end; { sin_ge0 }

global
function sin(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t       : Stagg[-1..Stagg_prec];
```

```
    d       : real;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN( t[1]+t[-1] ) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := sin( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      t := sin_ge0(t);
   if neg then t := -t;
   sin := t;
end; { sin }
```

1.10.7.1.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
{ y := sin(x) liefert Einschliessung von sin(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x : Stagg = ?');  read(x);
       y := sin(x);
       writeln('sin(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);   main;
end.
```

liefert für den Wertebereich $\sin(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
20     <enter>
x : Stagg = ?
20 Stagg components and 1 interval are needed in this sequence
1e-10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
sin(x) enthalten in:
 9.999999999999E-011
 9.999999833333E-024
 3.333333333333E-037
 3.341666666666E-050
 6.666666666646E-063
 8.253968253968E-076
 2.539682815255E-089
 7.319223985890E-102
 6.523067981401E-115
 3.147346480681E-128
 4.199175310286E-141
 4.213975317439E-154
 2.724657274921E-167
 8.251891869081E-180
 8.006099674322E-193
 5.107953166980E-206
 5.466133614046E-219
 5.223398483338E-232
 8.034131147377E-245
 0.000000000000E+000
[  7.887050303000E-258,  7.887050306000E-258 ]

Stagg_prec = ?
2      <enter>
x : Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-5,5]   <enter>
sin(x) enthalten in:
 0.000000000000E+000
 0.000000000000E+000
[ -1.000000000000E+000,  1.000000000000E+000 ]
```

Im nächsten Beispiel wählen wir ein Punktintervall `x`, dessen Element nur minimal kleiner ist als $2 \cdot 10^9\pi$.

```
 x =  6.283185307180E+009
     -4.135230747132E-004
     -3.344099423161E-017
      4.338798750212E-030
     -3.580501108154E-043
      1.563281257242E-056
     -2.002743930349E-069
     -3.157658640357E-082
     -3.826973435387E-096
      2.941876892191E-109
    +[1.164463450718E-123, 1.164463450718E-123]
```

ist zur Präzision $S = 10$ die größte punktförmige staggered Zahl, die kleiner ist als $2 \cdot 10^9\pi$. Im folgenden Programm wird `x` durch die Anweisungen

```
x := 2000000000*change_length(Pi_sm);
x := inf(x);
```

realisiert, wobei die Funktion `Pi_sm` vom Modul `stagg` exportiert wird und π mit maximaler Präzision `Stagg_prec=20` einschließt.
Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
    k    : integer;
 begin
    x := 2000000000 * change_length(Pi_sm);
    x := inf(x);
    y := sin(x);
    writeln('sin(x) enthalten in: ');
    write(y);
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\sin(x)$ mit `Stagg_prec=10` die folgende Einschließung:

```
sin(x) enthalten in:
 -8.162569622349E-136
 -5.682054038770E-151
 -4.221111928924E-164
 -5.897909860763E-177
 -9.288576219513E-190
 -3.186689225695E-203
 -1.296467573566E-216
 -3.305424038182E-229
 -9.129713384692E-242
  0.000000000000E+000
 [ -6.980000000020E-256, -6.959999999970E-256 ]
```

Das letzte Beispiel demonstriert die Leistungsfähigkeit der implementierten sin–Funktion! Obwohl das Argument `x` sehr dicht an der Nullstelle $2 \cdot 10^9\pi$ liegt, wird $\sin(x)$ noch mit einer Genauigkeit von 122 Dezimalstellen eingeschlossen.

1.10.7.2 $\cos(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen cos–Funktion:

$$\cos(x) := \{y \mid y = \cos(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\sin(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.7.2.1 Der Algorithmus

Da die sin-Funktion bereits implementiert ist, könnte man die cos-Funktion sehr einfach über die Funktionalgleichung $\cos(x) = \sin(x + \pi/2)$ realisieren. Da man jedoch nach dieser Methode in der Nähe der Nullstellen der cos-Funktion auch für Punktintervall–Argumente x starke Intervallüberschätzungen erhält, wählt man besser den folgenden Algorithmus[7]:

$$\text{Mit} \quad \zeta \in x,\ m := \mathrm{trunc}\left(\frac{\zeta}{\pi}\right) \quad \text{und} \quad y := x - (m + \frac{1}{2}) \cdot \pi \quad \text{gilt:}$$

$$\cos(x) = \begin{cases} \sin(y), & \text{falls } m \text{ ungerade} \\ -\sin(y), & \text{falls } m \text{ gerade;} \end{cases} \qquad \text{und} \quad y \Subset \left[-\frac{\pi}{2}, +\frac{\pi}{2}\right]$$

Das Symbol $\Subset$ bedeutet, daß das reduzierte Intervallargument y das Intervall $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ nach rechts oder links nur minimal überschreiten kann, wenn der relative Durchmesser von x und damit auch der relative Durchmesser von y hinreichend klein ist. Im Falle $0 \in y$ oder wenn der relative Durchmesser von y größer ist als 10^{-14}, wird $\sin(y)$ mit Hilfe der schon implementierten Intervall-sin-Funktion vorher im einfachen `real`–Format eingeschlossen, so daß die Auswertung von $\sin(y)$ im staggered Format nur mit Intervallen y erfolgt, deren relative Durchmesser kleiner sind als 10^{-14}. Bei Auswertung dieser $\sin(y)$-Funktion wird dabei natürlich auf die bereits implementierte staggered sin-Funktion zurückgegriffen.

[7] Die Funktion `trunc(...)` liefert den ganzzahligen Anteil ihres Arguments.

Die Berechnung des reduzierten Arguments y erfolgt mit der Präzision $S =$ `Stagg_prec` $= 20$, in der π durch `Pi_sm` im `real`–Format maximalgenau eingeschlossen wird. Beachten Sie bitte, daß jetzt für ein x in der Nähe einer cos-Nullstelle das reduzierte Intervall y direkt in die unmittelbare Nähe des Ursprungs transformiert wird, so daß im Gegensatz zur Funktionalgleichung $\cos(x) = \sin(x + \pi/2)$ jetzt ein zweite Argumentreduktion nicht notwendig ist! Dadurch erhält man mit dem jetzigen Algorithmus für z.B. $x \approx \pi/2$ eine sehr genaue Einschließung für $\cos(x)$, vergl. dazu das Beispiel auf Seite 121.

1.10.7.2.2 Der Programm-Quelltext

```
function cos(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-4;  { Argument > c:   -->  2. Argumentreduktion }
   c1   = 1e-14; { Rel. Durchmesser > c1 --> grobe Einschlg. }
   rlg3 = 2.096; { rlg3 > 1/log_10(3) }
var t,x1,y                : Stagg[-1..Stagg_prec];
   y1                     : Stagg[-1..20];
   N,S,k,j,p,pot          : integer;
   xi                     : interval;
   eps                    : real;
   negm,negy,reduce,testp : boolean;
begin
   t := x;  Normal(t);
   if t[1] > 6.746e9 then s_fu1_math_error(14);
   N := trunc(t[1]/PI_1);
   negm := N mod 2 = 0;  negy := false;  testp := false;
   { 1. Argumentreduktion in maximaler Praezision: }
   S := get_Stagg_prec;
   set_Stagg_prec(20);  { Maximale Praezision! }
   y1 := change_length(t);
   y1 := y1 - (N+0.5)*PI_sm; { 1. Argumentreduktion fertig! }
   reduce := abs(y1[1]) > c;
   { reduce = true  <===>  2. Arg.-Reduktion erforderlich }
   if (Sign(y1[-1]) <> Sign(y1[0])) and (Sign(y1[1])=0)
   then { 0 in y1 enthalten! } t := sin( ival(y1) ) else
   begin  { Null in y1 nicht enthalten! }
      negy := SIGN(y1[1]+y1[-1]) < 0;
      if negy then y1 := -y1; { Ab jetzt: inf(y1) > 0 }
      set_Stagg_prec(S);       x1 := change_length(y1);
      testp := x1[1] > 1.570796326794;
```

```
    if (x1[0]->x1[-1]) /< (x1[1]+x1[0]) > c1
       then t := sin( ival(y1) )  else
       begin  { Jetzt ein schmales Intervall: }
       if reduce then
       begin { 2. Arg.-Reduktion ist notwendig! }
          eps := rlg3 * (4 + log10(x1[1]));
          p := trunc(eps) + 1;
          j := round( power(3,p) );
          x1 := x1 / j;  { 2. Argument-Reduktion beendet }
       end;
       { Berechnung des Polynomgrades N und eine Oberschr. }
       xi := ival(x1); { eps des absoluten Approx.-Fehlers }
       sin_N_e(S,xi.sup,N,eps);
       { Polynomauswertung: }
       y := -x1*x1;  t := 1;
       For j := N downto 1 do
       begin
          k := 2*j;  pot := k*(k+1);
          t := t*y/pot + 1;
       end;
       t := x1 * t;       { t = Approximation fuer sin(x1) }
       t := t + Staggintval(-eps,eps);    { Approx-Fehler! }
       if reduce then   { Ruecktransformation: }
          For j := 1 to p do t := t*(3-4*(t*t));
    end;
  end;
  set_Stagg_prec(S);  { Alte Praezision! }
  if negm then   { Wahl des korrekten Vorzeichens: }
  begin
     if not negy then t := -t
  end     else if negy then t := -t;
  if testp then { Test auf:   -1 <= sin(x) <= +1 }
  begin
     y := Sup(t);
     if y > 1 then t := hull(Inf(t),Staggintval(1))
     else
        if Inf(t) < -1 then t := hull(Staggintval(-1),y);
  end;
  cos := t;
end; { cos }
```

1.10.7.2.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := cos(x);
       writeln('cos(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich $\cos(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
1e-6 0 0 0 0 0 0 0     <enter>
cos(x) enthalten in:
 9.999999999995E-001
 4.166666666666E-026
 5.277777777777E-039
 8.025793650793E-052
 6.480379188712E-065
 5.222546229490E-078
 0.000000000000E+000
[  6.739236476337E-091,  6.739236476343E-091 ]
```

```
Stagg_prec = ?
20     <enter>
x: Stagg = ?
20 Stagg components and 1 interval are needed in this sequence
1e9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   <enter>
cos(x) enthalten in:
 8.378871813639E-001
 2.334389775643E-015
 5.515723560595E-028
 7.694808811787E-041
 8.584601251106E-054
 8.202667780504E-067
 3.731147931698E-080
 9.185094390640E-093
 6.783462903187E-107
 5.589640128232E-120
 3.862394372478E-133
 9.531790677827E-147
 9.411577046133E-160
 1.079687896199E-173
 6.289046379585E-187
 5.017362913451E-200
 9.665946000917E-213
 9.082589769552E-226
 4.484847242514E-239
 0.000000000000E+000
[  2.136274960720E-252,  2.136275331259E-252 ]

Stagg_prec = ?
5      <enter>
x: Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0  <enter>
cos(x) enthalten in:
 9.999999999999E-001
 9.999999999999E-014
 9.999999999999E-027
 9.999999999999E-040
 9.999999999999E-053
[  9.999999999970E-066,  1.000000000000E-065 ]
```

Im nächsten Beispiel wählen wir ein Punktintervall x, dessen Element nur minimal kleiner ist als $\pi/2$.

```
x =  1.570796326795E+000
    -1.033807686783E-013
    -8.360248557901E-027
    -4.153003124471E-040
     1.048747229615E-053
     3.908203143104E-066
     4.993140174127E-079
    -2.894146600893E-092
     4.043256641153E-105
     3.235469223048E-118
  - [2.470888413733E-131, 2.470888413733E-131]
```

ist zur Präzision $S = 10$ die größte punktförmige staggered Zahl, die kleiner ist als $\pi/2$. Im folgenden Programm wird x durch die Anweisungen

```
x := change_length(Pih_sm);
x := inf(x);
```

realisiert, wobei die Funktion `Pih_sm` vom Modul `stagg` exportiert wird und $\pi/2$ mit maximaler Präzision `Stagg_prec=20` einschließt.
Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    x := change_length(Pih_sm);
    x := inf(x);
    y := cos(x);
    writeln('cos(x) enthalten in: ');
    write(y);  writeln;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\cos(x)$ mit `Stagg_prec=10` die folgende Einschließung:

```
cos(x) enthalten in:
 9.704064240558E-144
 7.251420513509E-157
 6.926055277982E-170
 2.311474477465E-183
 1.909822144054E-196
 8.783296672306E-209
 4.237824116893E-222
 3.915826356009E-235
 5.457282428346E-248
 0.000000000000E+000
[  1.730172000000E-261,  1.730178000000E-261 ]
```

Das letzte Beispiel demonstriert die Leistungsfähigkeit der implementierten cos–Funktion! Obwohl das Argument `x` sehr dicht an der Nullstelle $\pi/2$ liegt, wird $\cos(x)$ mit einer Genauigkeit von 123 Dezimalstellen eingeschlossen.

1.10.7.3 tan(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen tan–Funktion:

$$\tan(x) := \left\{ y \mid y = \tan(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty),\ \zeta \neq \frac{\pi}{2} + k\pi,\ k \in \mathbb{Z} \right\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg` und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\tan(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.7.3.1 Der Algorithmus

Zunächst wird der Wertebereich $\tan(x)$ mit Hilfe der schon implemntierten Intervall-tan–Funktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\tan(-x) \equiv -\tan(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\tan(x)$ eine garantierte Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Da die sin- und cos–Funktionen bereits implementiert sind, wählen wir die Funktionalgleichung

$$\tan(x) = \frac{\sin(x)}{\cos(x)} \tag{1.80}$$

Im Falle $0 \in \cos(x)$, erfolgt automatisch eine Fehlermeldung wegen einer Division durch 0, da dann das Intervall x eine Polstelle der tan–Funktion enthält.

Anmerkungen:

Die Einschließung von $\tan(x)$ läßt sich auch realisieren mit Hilfe der Potenzreihe:

$$\tan(x) = x \sum_{k=0}^{\infty} 2^{2k+2}(2^{2k+2} - 1) \cdot \frac{|B_{2k+2}|}{(2k+2)!} \cdot x^{2k}, \quad |x| < \frac{\pi}{2}, \tag{1.81}$$

wobei die B_{2k} die Bernoullisachen Zahlen sind.

Auf den ersten Blick scheint die Approximation mit Hilfe der abgebrochenen Potenzreihe in (1.81) günstiger zu sein als die zweimalige Auswertung der sin–Reihe in (1.80). Gegen die Auswertung von (1.81) sprechen aber die folgenden Punkte:

- Die Konvergenz der Reihe in (1.81) ist im Vergleich zur sin–Reihe um so schlechter, je größer $|x.\text{sup}| < \pi/2$ gewählt wird.

- Die Auswertung der abgebrochenen sin–Reihe erfolgt mit Hilfe der sehr einfachen Schleife von Seite 109. Eine so einfache Berechnung der tan–Reihe ist jedoch nicht möglich, da mir eine entsprechend einfache Rekursionsformel der Bernoulli–Zahlen nicht bekannt ist. Eine Alternative dazu wäre noch die Speicherung von Zähler und Nenner aller notwendigen rationalen Taylor–Koeffizienten aus (1.81). Bei den i.a. notwendigen hohen Polynomgraden erscheint mir der entsprechende Aufwand jedoch nicht vertretbar.

- Im Bereich $10^{-4} < x.\text{sup} < \pi/2$ wird eine zweite Argumentreduktion erforderlich, die z.B. mit Hilfe der Gleichung

  $$\tan(x) = \frac{\tan(x/3) \cdot (3 - \tan^2(x/3))}{1 - 3\tan^2(x/3)}$$

  durch p-malige Anwendung realisiert werden kann. Dabei wird man p so groß wählen, daß das reduzierte Argument $x/3^p$ die Bedingung $x.\text{sup}/3^p < 10^{-4}$ erfüllt. Ist dann t eine Einschließung von $\tan(x/3^p)$, so erhält man eine Einschließung von $\tan(x)$, wenn man die Gleichung

  $$f(t) := \frac{t \cdot (3 - t^2)}{1 - 3t^2} \tag{1.82}$$

  in einer Schleife p-mal durchläuft. Da bei der sin–Funktion die entsprechende Gleichung auf Seite 107

  $$f(t) := t \cdot (3 - 4t^2)$$

  sehr viel einfacher ist, wird besonders wegen der aufwendigen Division in (1.82) die Anwendung von (1.81) gegenüber (1.80) keinen gravierenden Laufzeitgewinn mehr liefern.

- Bei der tan–Funktion ist die Oberschranke des Betrags des relativen Approximationsfehlers gegeben durch:

$$\varepsilon(app) := \frac{8}{\pi^2} \cdot \frac{(2r/\pi)^{2N}}{(2r/\pi)^{-2} - 1} \cdot \frac{1}{1 - 2^{-2N-3}}, \quad |r| < \frac{\pi}{2}, \tag{1.83}$$

  wenn $\tan(r)$ approximiert wird durch:

$$\tan(r) \approx r \sum_{k=0}^{N} 2^{2k+2}(2^{2k+2} - 1) \cdot \frac{|B_{2k+2}|}{(2k+2)!} \cdot r^{2k}$$

  Nach (1.83) ist damit der Ausdruck für $\varepsilon(app)$ sehr viel komplizierter als der entsprechende Ausdruck für die sin–Funktion:

$$\varepsilon(app) := \frac{r}{\sin(r)} \cdot \frac{r^{2N+2}}{(2N+3)!},$$

  so daß auch die Berechnung des Polynomgrades N und des Approximationsfehlers bei der tan–Funktion sehr viel aufwendiger ist als bei der sin–Funktion. Dies fällt allerdings weniger stark ins Gewicht, da die entsprechenden Berechnungen beider Größen im einfachen `real`–Format auszuführen sind.

Zusammenfassend kann man sagen, daß der Vorteil der einfach zu programmierende Gleichung (1.80) den nur geringen Laufzeitnachteil mehr als kompensieren wird.

1.10.7.3.2 Der Programm-Quelltext

```
global function tan(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
    d        : real;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
```

```
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := tan( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      t := sin(t) / cos(t);
   if neg then t := -t;
   tan := t;
end; { tan }
```

1.10.7.3.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := tan(x);
       writeln('tan(x) enthalten in: ');
       write(y);  writeln;
    until true
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\tan(x)$ die folgenden garantierten Einschließungen:

```
Stagg_prec = ?
1      <enter>
x: Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 [0,0]      <enter>
tan(x) enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
tan(x) enthalten in:
 1.557407724654E+000
 9.022305069748E-013
 7.458360173087E-027
 2.507723815200E-040
 3.838394660569E-053
 8.861397151727E-066
 2.895550999652E-079
 2.242983804633E-093
 8.214117481666E-106
 1.332355461812E-119
 4.558937606071E-132
 6.845489044392E-145
 9.358604316714E-158
 7.908036824613E-171
 2.747069555973E-184
 4.164061077553E-197
 5.247302506796E-210
 8.505070413523E-223
[  8.514491762050E-236,  8.514491762230E-236 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1.570796326794 8.96619231322e-13 0 0 0 0 0   <enter>
tan(x) enthalten in:
-3.242960156753E+024
-8.885193284787E+011
-7.884024922030E-003
-4.398377781308E-016
-9.321679147733E-029
-5.530917909312E-042
[ -3.906596121106E-055, -3.906596121064E-055 ]
```

1.10.7.4 $\cot(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen cot–Funktion:

$$\cot(x) := \{y \,|\, y = \cot(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty),\ \zeta \neq k\pi,\ k \in \mathbb{Z}\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg` und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\cot(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.7.4.1 Der Algorithmus

Zunächst wird der Wertebereich $\cot(x)$ mit Hilfe der schon implemntierten Intervall-cot–Funktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. In allen anderen Fällen wird $\cot(x)$ realisiert mit der Funktionalgleichung

$$\cot(x) := \frac{\cos(x)}{\sin(x)}$$

1.10.7.4.2 Der Programm-Quelltext

```
function cot(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t    : Stagg[-1..Stagg_prec];
    d    : real;
   grob : boolean;
begin
   t := x;  Normal(t);
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := cot( ival(t) )
```

```
   else  { t ist schmal: }
      t := cos(t) / sin(t);
   cot := t;
end; { cot }
```

1.10.7.4.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := cot(x);
       writeln('cot(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für $\cot(x)$ die folgenden garantierten Einschließungen:

```
Stagg_prec = ?
3      <enter>
x: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
1.570796326794 8.96619231322e-13 0 0   <enter>
cot(x) enthalten in:
-3.083602485579E-025
-1.415300312447E-039
-8.951252770384E-053
[ -6.091796867446E-066, -6.091796865891E-066 ]
```

```
Stagg_prec = ?
12     <enter>
x: Stagg = ?
12 Stagg components and 1 interval are needed in this sequence
1e-100 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
cot(x) enthalten in:
 9.999999999999E+099
 9.999999999999E+086
 9.999999999999E+073
 9.999999999999E+060
 9.999999999999E+047
 9.999999999999E+034
 9.999999999999E+021
 9.999999999999E+008
 9.999999999999E-005
 9.999999999999E-018
 9.999999999999E-031
 9.999999999999E-044
[  9.999999999877E-057,  1.000000000021E-056 ]

Stagg_prec = ?
10     <enter>
x: Stagg = ?
10 Stagg components and 1 interval are needed in this sequence
3.141592653589 7.93238462643e-13 0 0 0 0 0 0 0 0 0     <enter>
cot(x) enthalten in:
-2.609062035603E+024
-1.320801818902E+011
-9.218578567661E-002
-5.858849729767E-015
-3.947424669608E-028
-9.915588794176E-041
-9.235754740153E-054
-9.152208767223E-067
-8.950289358197E-080
-3.293503828797E-093
[ -2.769958964690E-106, -2.769958964210E-106 ]
```

1.10.8 Die inversen trigonometrischen Funktionen

1.10.8.1 arcsin(x)

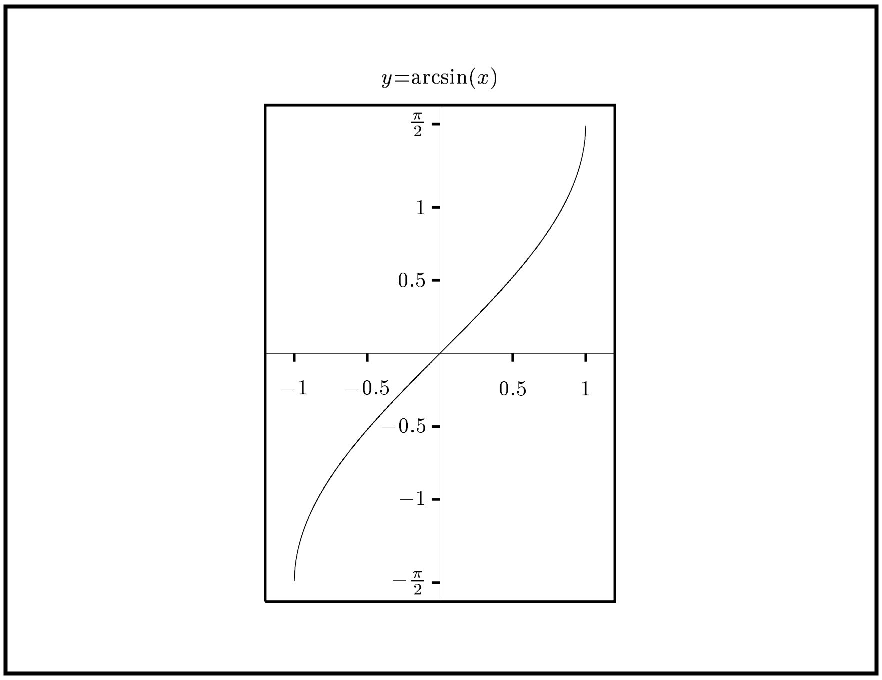


Abbildung 1.1: Graph der arcsin–Funktion

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arcsin–Funktion:

$$\arcsin(x) := \{y \,|\, y = \arcsin(\zeta),\ \zeta \in x \subseteq [-1, +1]\}$$

Im Programm wird x realisiert durch die Variable `x` vom Typ `Stagg`, und es gilt: $x_j = \mathtt{x[j]}$, wobei $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß.

Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von arcsin(x) erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

In einer Diplomarbeit von Frederic Toussaint am Institut für Angewandte Mathematik der Universität Karlsruhe (1992) wird unter `C-XSC` die arcsin-

Funktion implementiert mit Hilfe der arctan-Funktion über die Funktionalgleichung

$$\arcsin(x) = \arctan\left(\frac{x}{\sqrt{1-x^2}}\right), \quad x \subset [-1, +1],$$

wobei die Argumentreduktion realisiert wird durch:

$$\arctan(x) = 2^m \cdot \arctan(s_m), \quad \text{mit:} \quad s_0 := x, \; s_{m+1} := \frac{s_m}{1 + \sqrt{1 + s_m^2}}$$

In der Linux-Umgebung meines Rechners ist es jedoch insbesondere für größere `Stagg_prec`-Parameter laufzeitmäßig günstiger, wenn man zunächst die arcsin-Funktion mit dem nachfolgenden Algorithmus realisiert und dann die arctan-Funktion berechnet mit:

$$\arctan(x) = \arcsin\left(\frac{x}{\sqrt{1+x^2}}\right)$$

1.10.8.1.1 Der Algorithmus

Zunächst wird $\arcsin(x)$ im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen $\arcsin(-x) = -\arcsin(x)$ kann man sich beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\arcsin(x)$ eine garantierte Einschließung zu berechnen, wenn gilt

$$0 < \inf(x) \leq \sup(x) \leq 1, \quad \text{mit:} \quad \operatorname{diam}(x)/\inf(x) \leq 10^{-14}$$

Argumentreduktion:

Da das Argumentintervall x dicht bei 1 liegen kann, wäre in einem solchen Fall die Approximation durch ein Taylor-Polynom viel zu aufwendig, so daß vorher eine Argumentreduktion auf ein staggered Intervall t erforderlich ist, das folgende Eigenschaft haben sollte:

$$0 < \inf(t) \leq \sup(t) \ll +1$$

Dabei ist jedoch zu beachten, daß eine solche Argumentreduktion ohne die Speicherung zusätzlicher Daten durchgeführt werden muß, da solche Daten erfahrungsgemäß mit großem Aufwand in der gleichen Präzision bereitgestellt werden müssen, in der die staggered Arithmetik durchgeführt werden soll. Man findet die gewünschte Argumentreduktion auf ein solches Intervall t mit sogar beliebig kleinem Durchmessers nach folgenden Überlegungen:

Mit Hilfe der in $r \in [-1, 1]$ gültigen Beziehungen:

$$\arcsin(r) = \frac{\pi}{2} - \arcsin\sqrt{1-r^2} = \frac{\pi}{2} - 2\cdot\arcsin\sqrt{\frac{1-r}{2}}$$

erhält man unmittelbar die folgende Identität:

$$\arcsin(r) = 2\cdot\arcsin\sqrt{0.5\cdot(1-\sqrt{1-r^2})}, \quad r \in [0, +1] \tag{1.84}$$

Setzt man anschließend $u(r) := \sqrt{0.5\cdot(1-\sqrt{1-r^2})}$ und wendet man in $\arcsin(r) = 2\cdot\arcsin(u)$ auf $\arcsin(u)$ die Gleichung (1.84) an, so folgt:

$$\arcsin(r) = 4\cdot\arcsin\sqrt{0.5 - \sqrt{0.125(1+\sqrt{1-r^2}\ )}}$$

Diese Schritte lassen sich beliebig oft wiederholen und liefern das folgende Ergebnis:

$$g_1(r) := \sqrt{0.125\cdot(1+\sqrt{1-r^2})}, \qquad z(n) := 2^{n+1}$$

$$\text{Iteration:}\quad g_{k+1}(r) = \sqrt{0.125\cdot(1+\sqrt{0.5+g_k(r)}\ )},\ k = 1, 2, \ldots, z(n)-1$$

$$t_n(r) := \sqrt{\frac{1}{2} - g_{z(n)}(r)} \tag{1.85}$$

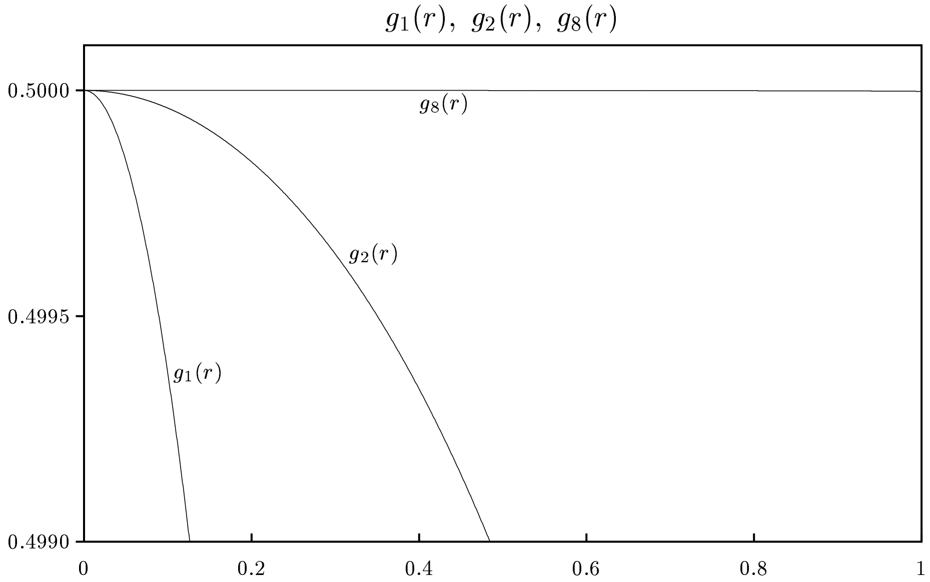
$$\arcsin(r) = 4^{z(n)}\cdot\arcsin(\,t_n(r)\,), \quad r \in [0, 1], \quad n = 0, 1, 2, \ldots \tag{1.86}$$

Bezüglich der $g_k(r)$ gilt folgendes:

1. Mit $g_1(r)$ sind auch alle $g_k(r)$ monoton fallend in $[0, 1]$, da die Wurzel aus einer positiven und monoton fallenden Funktion selbst wieder eine monoton fallende Funktion ist.
2. $g_k(r) \le g_k(0) = 1/2, \quad r \in [0, +1], \quad k = 1, 2, \ldots$
3. $g_{k+1}(r) \ge g_k(r), \quad r \in [0, +1];$ der Beweis folgt aus der äquivalenten Aussage $\sqrt{1/2 + g_k} \ge 8g_k^2 - 1$ mit Hilfe von $g_k(r) \le 1/2$.

Beachten Sie bitte, dass man beim Startwert $g_1(r)$ Auslöschung vermeidet, wenn $\sqrt{1-r^2} \equiv \sqrt{(1-r)\cdot(1+r)}$ benutzt wird. Die Berechnung der iterierten Werte $g_{k+1}(r)$ erfolgt dann numerisch stabil ohne jede Auslöschung!

Die Graphen der Funktionen $g_1(r), g_2(r)$ und $g_8(r)$ findet man in folgender Abbildung:



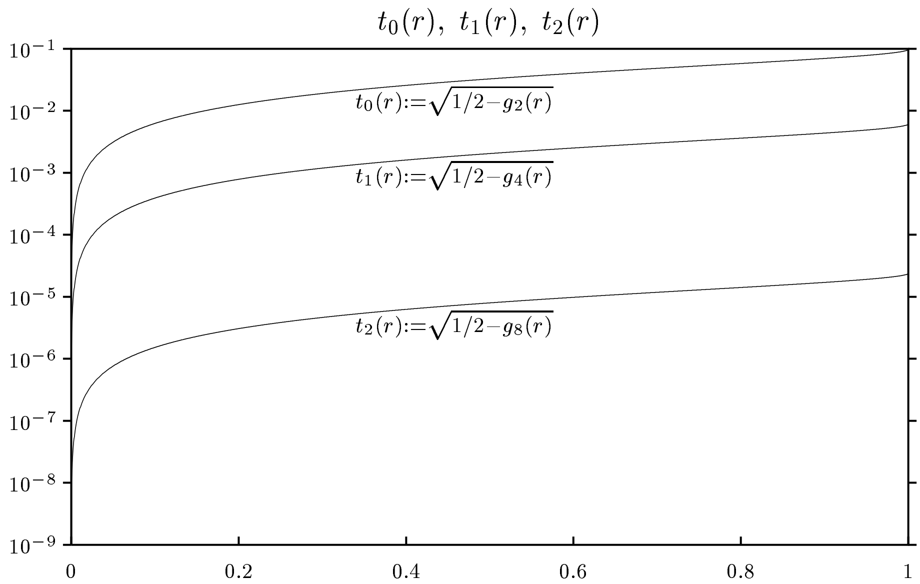
Wegen $g_8(1) \approx 0.5$ erscheint in obiger Abbildung die monoton fallende Funktion $g_8(r)$ als konstante Funktion. Die oben angegebenen Ergebnisse für die $g_k(r)$ lassen sich auf Grund der Beziehung (1.85) direkt auf die $t_n(r)$ übertragen:

1. $t_n(r) := \sqrt{1/2 - g_{z(n)}(r)}$ ist monoton wachsend in $[0, 1]$
2. $0 \le t_n(r) \le t_n(1), \quad n = 0, 1, 2, \ldots$
3. $t_{n+1}(r) \le t_n(r), \quad x \in [0, 1]$

Die monotonen Funktionen $t_n(r)$ liefern nach (1.85) eine Abbildung des Intervalls $[0, 1]$ nach $[0, t_n(1)]$, und wegen der Beziehung $t_{n+1}(1) < t_n(1)$ läßt sich damit für ein hinreichend großes n zu einem gegebenem Intervallargument $x \subset [0, +1]$ ein hinreichend schmales Intervall $t_n(x)$ angeben, in dem zur Berechnung der arcsin–Funktion nach (1.86) nur noch wenige Intervall–Summanden der Taylorreihe zu addieren sind. Unter $t_n(x)$ verstehen wir den Wertebereich der Funktion $t_n(r)$:

$$t_n(x) := \{y \mid y = t_n(r),\ r \in x\}$$

Die Graphen von $t_0(r), t_1(r), t_2(r)$ findet man für $r \in [0, +1]$ in folgender Abbildung:



Zu gegebenem Intervallargument $x \subseteq [0, +1]$ sei nun $t_n(x)$ der Wertebereich der Funktion $t_n(r)$, mit $r \in x$. Ist dann $\arcsin(x)$ der entsprechende Wertebereich der arcsin–Funktion, so gilt nach (1.86) von Seite 133

$$\arcsin(x) = 4^{z(n)} \cdot \arcsin(\, t_n(x)\,), \quad x \subset [0, 1], \tag{1.87}$$

Beachten Sie bitte, daß in (1.87) auf beiden Seiten des Gleichheitszeichens zwei identische Wertebereiche stehen!

Die eigentliche Aufgabe besteht nun darin, für die obigen Wertebereiche $\arcsin(x)$ bzw. $\arcsin(\, t_n(x)\,)$ mit Hilfe eines geeigneten Taylorpolynoms eine möglichst enge Einschließung zu berechnen. Im Fall $\sup(x) < 10^{-3}$ gilt dann $\arcsin(x) \approx P_N(x)$, wobei $P_N(x)$ ein geeignetes Taylor–Polynom ist. Gilt jedoch $\sup(x) \geq 10^{-3}$, so wird zunächst das Argument x auf das Intervall $t_n(x)$ reduziert und dann die Approximation $\arcsin(t_n(x)) \approx P_N(t_n(x))$ durchgeführt. Der Polynomgrad N ist jeweils so zu wählen, daß die durch $S :=$ `Stagg_prec` vorgegebene Genauigkeit erreicht wird. Genauere Einzelheiten findet man dazu auf Seite 138.

Bei der Argumentreduktion ist es sinnvoll, bei kleinerem S die Funktion $t_1(r) := \sqrt{1/2 - g_4(r)}$ zu wählen und für große S-Werte die Reduktion mit $t_2(r) := \sqrt{1/2 - g_8(r)}$ vorzunehmen. Die Argumentreduktion erfolgt damit nach folgendem Schema:

```
if sup(x) < 1e-3 then arcsin(x) ≈ P_N(x) else
   if S < 6 then
   begin
      t_1(x) := sqrt(1/2 - g_4(x));      { t_1(1) < 6.136 · 10^-3 }
      arcsin(t_1(x)) ≈ P_N(t_1(x))
   end      else
   begin
      t_2(x) := sqrt(1/2 - g_8(x));      { t_2(1) < 2.397 · 10^-5 }
      arcsin(t_2(x)) ≈ P_N(t_2(x))
   end;
```

Beachten Sie bitte, daß die obigen Intervalle $\arcsin(x), P_N(x), t_1(x), g_4(x)$ usw. alles Wertebereiche sind. $P_N(x)$ ist beispielsweise definiert durch

$$P_N(x) := \left\{ y \,|\, y = P_N(r) := \sum_{k=0}^{N} q_k \cdot r^{2k+1},\ r \in x \right\}$$

und nicht zu verwechseln mit der Intervall-Auswertung dieses Polynoms:

$$\mathtt{P}_N(x) := \sum_{k=0}^{N} \mathtt{q}_k \diamond x^{2k+1}, \quad \text{es gilt jedoch stets:} \quad P_N(x) \subseteq \mathtt{P}_N(x);$$

Die Einschließung des Wertebereichs $P_N(x)$ durch das Ergebnis $\mathtt{P}_N(x)$ der Intervallauswertung von $P_N(r)$ ist nicht auf Polynome beschränkt, sondern gilt ganz allgemein für jede stetige Funktion $f(r)$, deren Intervallauswertung definiert ist.[8] Den entsprechenden Satz findet man in [2, Seite 31]. Bei den folgenden Überlegungen beziehen wir uns jedoch zunächst nur auf die Wertebereiche der auftretenden Funktionen! In den nächsten Abschnitten berechnen wir zunächst eine Einschließung für $\arcsin(r)$ mit $r \in (0, +1]$, wobei $\arcsin(r)$ durch ein Taylor-Polynom $P_N(r)$ approximiert wird. Danach läßt sich dann eine Einschließung des Wertebereichs $\arcsin(x)$ angeben, und

[8] Bei der Intervallauswertung von $f(r)$ wird im Funktionsterm von f zunächst r durch x ersetzt und alle Rechenoperationen werden als Intervalloperationen interpretiert. Kommt dann im Funktionsterm die Variable x mehrfach vor, so kann es zu deutlichen Intervallüberschätzungen kommen, weil bei den Intervalloperationen angenommen wird, daß die Elemente $r \in x$ in jeder Variablen x **unabhängig** voneinander gewählt werden. Durch solche Intervallüberschätzungen können Definitionsbereiche überschritten werden, oder es kann zur Division durch ein Intervall kommen, das die Zahl Null enthält, so daß dann die Intervallauswertung von f nicht mehr definiert ist. Um die Überschreitung des Definitionsbereichs der Wurzelfunktion zu vermeiden, wird bei der Auswertung von $\mathtt{g}_1(x)$ der Wertebereich $\sqrt{1-x^2}$ eingeschlossen durch die bereits implementierte Funktion `sqrt1mx2(x)`.

im letzten Schritt wird daraus der Algorithmus entwickelt, mit dem $\arcsin(x)$ auf einem Rechner in der BCD–Version von **PASCAL-XSC** eingeschlossen wird.

Reihenentwicklung:

Für die arcsin-Funktion existiert die folgende Potenzreihe:

$$\arcsin(r) = \sum_{k=0}^{\infty} q_k \cdot r^{2k+1}, \quad |r| < 1;$$

Die Koeffizienten q_k sind gegeben durch:

$$\begin{aligned}
q_0 &:= 1 \\
q_1 &:= \frac{1}{2\cdot 3} \\
q_2 &:= \frac{1\cdot 3}{2\cdot 4\cdot 5} \\
q_3 &:= \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7} \\
q_4 &:= \frac{1\cdot 3\cdot 5\cdot 7}{2\cdot 4\cdot 6\cdot 8\cdot 9}, \quad \ldots \\
q_k &:= \frac{(2k)!}{(2^k k!)\cdot(2k+1)} = \frac{1}{2k+1}\prod_{j=1}^{k}\frac{2j-1}{2j}, \qquad k = 1, 2, \ldots \\
q_{k+1} &:= q_k \cdot \frac{(2k+1)\cdot(2k+1)}{(2k+2)\cdot(2k+3)}, \qquad k = 0, 1, 2, \ldots
\end{aligned}$$

Mit Hilfe der letzten Rekursionsformel lassen sich die Koeffizienten q_k im Programm bequem einschließen. Im Bereich $0 \le r < 1$ gilt:

$$\begin{aligned}
&(1.88) \quad P_N &=&\; P_N(r) := \sum_{k=0}^{N} q_k \cdot r^{2k+1} \le \arcsin(r), \quad N = 0, 1, \ldots \\
&(1.89) \quad P_N &=&\; P_{N-1} + q_N \cdot r^{2N+1}, \qquad N = 1, 2, \ldots
\end{aligned}$$

Die Funktion $\arcsin(r)$ und das Taylorpolynom $P_N(r)$ sind dabei monoton wachsend in r. Zur Einschließung des Wertebereichs $\arcsin(x)$ werden wir diese Eigenschaft auf Seite 139 ganz wesentlich benutzen. Mit Hilfe der Rekursionsformel (1.89) wird im Programm der Wertebereich $P_N(x)$ des Taylor-Polynoms sehr effektiv eingeschlossen.

Approximationsfehler

Im Bereich $0 < r < +1$ gilt für den relativen Approximationsfehler $\varepsilon_N(r)$ die Abschätzung:

$$\begin{aligned}
|\varepsilon_N(r)| &:= \left| \frac{\arcsin(r) - r \cdot \sum_{k=0}^{N} q_k \cdot r^{2k}}{\arcsin(r)} \right| = \frac{\arcsin(r) - P_N(r)}{\arcsin(r)} \\
&= \left| \frac{r}{\arcsin(r)} \right| \sum_{k=N+1}^{\infty} q_k \cdot r^{2k} \leq \sum_{k=N+1}^{\infty} q_k \cdot r^{2k} \\
&= q_{N+1} \cdot r^{2N+2} \sum_{k=N+1}^{\infty} \frac{q_k}{q_{N+1}} \cdot \frac{r^{2k}}{r^{2N+2}} \\
&= q_{N+1} \cdot r^{2N+2} \left\{ 1 + \underbrace{\frac{q_{N+2}}{q_{N+1}}}_{\in(0,1)} \cdot r^2 + \underbrace{\frac{q_{N+3}}{q_{N+1}}}_{\in(0,1)} \cdot r^4 + \ldots \right\} \\
&< q_{N+1} \cdot r^{2N+2} \sum_{k=0}^{\infty} r^{2k} = q_{N+1} \cdot r^{2N+2} \cdot \frac{1}{1-r^2} =: A_N(r);
\end{aligned}$$

und mit

$$P_N(r) = \arcsin(r) \cdot (1 + \varepsilon_N(r)), \qquad |\varepsilon_N(r)| < A_N(r)$$

erhält man dann für $\arcsin(r)$ die Einschließung:

$$(1.90) \qquad P_N(r) \leq \arcsin(r) = \frac{P_N(r)}{1 - \varepsilon_N(r)} < \frac{P_N(r)}{1 - A_N(r)} =: \alpha_N(r)$$

Bevor wir $\alpha_N(r)$ weiter nach oben abschätzen, um die aufwendige Division zu vermeiden, benötigen wir zunächst eine Oberschranke von $A_N(r)$, die für jeden Wert von $\texttt{Stagg\_prec} \geq 1$ gelten soll. Da im staggered Format mit mindestens[9] $13 \cdot (\texttt{Stagg\_prec} + 1)$ Dezimalstellen gerechnet wird, ist es sinnvoll, für den Approximationsfehler folgende Oberschranke zu verlangen:

$$(1.91) \qquad A_N(r) \leq 10^{-13 \cdot (\texttt{Stagg\_prec}+1)} \leq 10^{-26}$$

[9]Es wird vorausgesetzt, daß sich die staggered Komponenten nicht überlappen.

Bei der Implementierung der Funktion `arcsin_ge0` wird also die Abschätzung $A_N(r) \leq 10^{-26}$ garantiert, d.h. zu gegebenem r wird N schrittweise so lange um 1 erhöht, bis die erste Bedingung in (1.91) erfüllt ist. Unter dieser Voraussetzung gilt dann mit $\quad \delta := 1 + 10^{-12} \quad$ die Abschätzung:

$$\frac{1}{1 - A_N(r)} \leq 1 + \frac{A_N(r)}{1 - 10^{-26}} \leq 1 + \delta \cdot A_N(r)$$

Zusammen mit der Abschätzung (1.90) erhält man damit für $\arcsin(r)$ die folgende Einschließung:

$$A_N(r) := q_{N+1} \cdot r^{2N+1} \cdot \frac{r}{1 - r^2} \leq 10^{-26}, \quad \delta := 1 + 10^{-12}; \tag{1.92}$$

$$P_N(r) \leq \arcsin(r) \leq P_N(r) \cdot [1 + \delta \cdot A_N(r)] \tag{1.93}$$

Damit haben wir eine Einschließung von $\arcsin(r)$ gefunden, in der die Oberschranke ohne zeitaufwendige Division berechnet werden kann.

Einschließung des Wertebereichs $\arcsin(x)$**:**

Mit Hilfe von (1.93) sind wir nun in der Lage, für ein staggered Intervall–Argument t, das durch eine mögliche Argumentreduktion nach Seite 136 im ungünstigsten Fall beschränkt ist auf

$$t \subseteq [0,\, 6.136 \cdot 10^{-3}]$$

eine Einschließung des Wertebereichs $\arcsin(t)$ anzugeben. Da $P_N(r)$ und $A_N(r)$ monoton wachsende Funktionen in $r \in t$ sind, folgt nach (1.93)

$$\arcsin(t) \subseteq P_N(t) \cdot [1, 1 + \delta \cdot \sup(\, A_N(t)\,)] \tag{1.94}$$

Beachten Sie bitte, daß in (1.94) $P_N(t)$ und $A_N(t)$ die Wertebereiche der reellen Funktionen $P_N(r)$ und $A_N(r)$ bedeuten.

Einschließung des Wertebereichs $\arcsin(x)$ **auf der Maschine:**

Ist x das vorgegebene staggered Intervallargument und $t_n(x)$ der Wertebereich der in (1.85) definierten Funktion $t_n(r)$, so gilt nach (1.94):

$$\arcsin(t_n(x)) \subseteq P_N(t_n(x)) \cdot [1, 1 + \delta \cdot \sup(\, A_N(t_n(x))\,)] \tag{1.95}$$

Wertet man nun $t_n(r)$ für $r \in x$ intervallmäßig aus[10] und erhält man dabei auf der Maschine das staggered Intervall $\mathtt{t}$, so gilt nach [2, Seite 31] $t_n(x) \subseteq \mathtt{t}$ und damit nach (1.95):

$$\arcsin(t_n(x)) \subseteq P_N(\mathtt{t}) \cdot [1, 1 + \delta \cdot \sup(\, A_N(\mathtt{t})\,)] \tag{1.96}$$

[10] Vergleichen Sie dazu die Fußnote von Seite 136.

Ist nun das staggered Intervall `P` das Maschinenergebnis der Intervallauswertung von $P_N(r)$ für $r \in \mathtt{t}$ nach (1.89) und bedeutet entsprechend `A` das Ergebnis der Intervallauswertung[11] von $A_N(r)$ nach (1.92), so gilt wieder

$$P_N(\mathtt{t}) \subseteq \mathtt{P} \qquad \text{und} \qquad A_N(\mathtt{t}) \subseteq \mathtt{A} \tag{1.97}$$

Definiert man das staggered Intervall `e : Stagg[-1..1]` wie folgt:

```
e[1] := 1;    e[-1] := 0;    e[0] := delta *> A.sup;
```

so gilt: $[1, 1+\delta\cdot\sup(\,A_N(\mathtt{t})\,)] \subseteq \mathtt{e}$, und mit (1.95) und (1.86) folgt schließlich:

$$\arcsin(t_n(x)) \subseteq \mathtt{P} \diamond \mathtt{e} \tag{1.98}$$

$$\arcsin(x) \subseteq 4^{z(n)} \diamond \mathtt{P} \diamond \mathtt{e} \tag{1.99}$$

Damit sind alle wichtigen Schritte skizziert, die bei der Implementierung der `arcsin` Funktion zu beachten sind. Wegen $P_N(\mathtt{t}) \subseteq \mathtt{P}$ ist die Abschätzung irgendeines Polynom–Auswertefehlers überflüssig, so daß die Realisierung der arcsin–Funktion im staggered Format sehr viel einfacher ist als für die Datentypen real oder Longreal. Den Quelltext der arcsin–Funktion findet man für die BCD–Version von **PASCAL-XSC** im folgenden Abschnitt.

1.10.8.1.2 Der Programm-Quelltext

```
function arcsin_ge0(var x : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet arcsin(x), falls x >= 0;  diese Bedingung wird }
{ in dieser Funktion jedoch nicht abgefragt!               }
{ Blomquist, 11.03.2001;                                   }
const delta = 1.000000000001;
var r,rr,rpot,P,q_n  : Stagg[-1..Stagg_prec];
   e                 : Stagg[-1..1];
   P1,r1             : Stagg;
   RelErr            : real;
   reduc             : boolean;
   ir,A_n,iq_n,irpot : interval;
   n,S,k             : integer;
   fac,z             : array[1..2] of integer;
begin
   z[1]   := 3;    z[2]   := 7;
```

[11] Aus Laufzeitgründen erfolgt die Intervallauswertung von $A_N(r)$ im einfachen `real`-Format.

```
   fac[1] := 256;  fac[2] := 65536;
   RelErr := comp(0.1,-13*(Stagg_prec+1)+1);
   r := x;  { x bleibt erhalten }
   reduc := r[1] > 1e-3;
   if reduc then
   begin  { Argumentreduktion auf: [0,6.136e-3] }
      S := Stagg_prec;
      if S < 6 then k := 1 else k := 2;
      allocate(P1,-1..S+1);   allocate(r1,-1..S+1);
      set_Stagg_prec(S+1);
      r1 := change_length(r);
      P1 := sqrt(0.125 * (1 + sqrt1mx2(r1)));
      for n := 1 to z[k] do P1 := sqrt(0.125*(1+sqrt(0.5+P1)));
      r1 := sqrt(0.5-P1); { --> r1 in [0,6.136e-3] }
      set_Stagg_prec(S);
      r := change_length(r1);  free(r1);  free(P1);
   end;
   { Es gilt: 0 <= r.sup <= 6.136*10^-3 }
   rr := r * r;
   rpot := r;
   ir := r;  ir := ir/(1-ir*ir);
   P := rpot;  { P=q_0*r }
   n := 1;
   q_n := 1; q_n := q_n/6; { = q_1 = 1/(2*3) }

   repeat
      rpot := rpot * rr;
      P := P + q_n * rpot;
      e := (2*n+2)*(2*n+3);  r := (2*n+1)*(2*n+1);
      q_n := q_n*r/e;
      n := n + 1;
      iq_n := q_n;
      irpot := rpot;
      A_n := iq_n * irpot * ir;
   until A_n.sup < RelErr;
   if reduc then P := fac[k] * P;
   e[1] := 1;  e[-1] := 0;  e[0] := delta *> A_n.sup;
   P := P * e;  { ---> arcsin(x) in s enthalten }
   arcsin_ge0 := P;
end; { arcsin_ge0 }
```

```
global function arcsin(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
    d        : real;
   neg,grob : boolean;
begin
   if Stagg_prec > 19 then s_fu1_math_error(12);
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := arcsin( ival(t) )  { ival(t) : interval }
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      t := arcsin_ge0(t);
   if neg then t := -t;
   arcsin := t;
end; { arcsin }
```

1.10.8.1.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm der BCD–Version

```
program tests;
{ y := arcsin(x) liefert Einschliessung von arcsin(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x : Stagg = ?');  read(x);
       y := arcsin(x);
       writeln('arcsin(x) enthalten in: ');
       write(y);  writeln;
    until false;
```

```
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man für $\arcsin(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
4      <enter>
x : Stagg = ?
4 Stagg components and 1 interval are needed in this sequence
0.96875 0 0 0 [0,0]   <enter>
arcsin(x) enthalten in:
 1.320140664458E+000
 7.658254821877E-013
 1.892558200600E-026
 2.289246010208E-039
[  1.600195442340E-052,  1.600195442344E-052 ]

Stagg_prec = ?
11     <enter>
x : Stagg = ?
11 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0   <enter>
arcsin(x) enthalten in:
 1.570796326794E+000
 8.966192313216E-013
 9.163975144209E-026
 8.584699687552E-039
 9.104874722961E-052
 5.390820314310E-065
 4.499314017412E-078
 6.710585339910E-091
 7.404325664115E-104
 3.323546922304E-117
 7.752911158626E-130
[  7.970406424055E-143,  7.970406424057E-143 ]
```

```
Stagg_prec = ?
19     <enter>
x : Stagg = ?
19 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 [-1e-267,0]      <enter>
arcsin(x) enthalten in:
 1.570796326794E+000
 8.966192313216E-013
 9.163975144209E-026
 8.584699687552E-039
 9.104874722961E-052
 5.390820314310E-065
 4.499314017412E-078
 6.710585339910E-091
 7.404325664115E-104
 3.323546922304E-117
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  7.752271706580E-130,  7.753091984913E-130 ]

Stagg_prec = ?
5      <enter>
x : Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
-0.75 0 0 0 0 0    <enter>
arcsin(x) enthalten in:
-8.480620789814E-001
-8.100805294433E-014
-8.998418080073E-027
-3.662132631126E-040
-4.286071816357E-053
[ -2.008212284853E-067, -2.008212284619E-067 ]
```

1.10.8.2 arccos(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arccos–Funktion:

$$\arccos(x) := \{y \,|\, y = \arccos(\zeta),\ \zeta \in x \subset [-1, +1]\}$$

Im Programm wird x realisiert durch die Variable `x` vom Typ `Stagg`, und es gilt: $x_j =$ `x[j]`, wobei $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß.

Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\arctan(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.8.2.1 Der Algorithmus

Die Einschließung des Wertebereichs $\arccos(x)$ erfolgt mit den Funktionalgleichungen:

$$\arccos(x) = \begin{cases} \pi - \arcsin(\sqrt{1 - x^2}) & \text{falls } \sup(x) < -0.75 \\ \arcsin(\sqrt{1 - x^2}) & \text{falls } \inf(x) > +0.75 \\ \pi/2 - \arcsin(x) & \text{sonst} \end{cases}$$

1.10.8.2.2 Der Programm-Quelltext

```
function arccos(var x : Stagg) : Stagg[-1..Stagg_prec];
const c =  0.75;
var t : Stagg[-1..Stagg_prec];
begin
   t := x;  Normal(t);
   if t[1] > c then t := arcsin( sqrt1mx2(t) ) else
      if t[1] < -c then t := Pi_sm - arcsin( sqrt1mx2(t) )
      else t := Pih_sm - arcsin(t);
   arccos := t;
end;
```

Beachten Sie bitte, daß der Prozeduraufruf `Normal(t)` die Einschließung $x \subseteq$ `t` realisiert und daß sich die von Null verschiedenen `t`$[j]$ für $j = 1, 2, \ldots$ nach dem Prozeduraufruf nicht mehr überlappen.

1.10.8.2.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
{ y := arccos(x) liefert Einschliessung von arccos(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x : Stagg = ?');  read(x);
       y := arccos(x);
       writeln('arccos(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man für arccos(x) die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x : Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0 0   <enter>
arccos(x) enthalten in:
 1.570796326794E+000
 8.966192313216E-013
 9.163975144209E-026
 8.584699687552E-039
 9.104874722961E-052
 5.390820314310E-065
[  4.499314017412E-078,  4.499314017413E-078 ]
```

```
Stagg_prec = ?
5
x : Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0
arccos(x) enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]

Stagg_prec = ?
19
x : Stagg = ?
19 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1e-267
arccos(x) enthalten in:
 4.472135954999E-134
 5.793928183473E-147
 3.746255247088E-160
 1.236719223051E-173
 4.485417944908E-186
 2.104185127560E-199
 9.798828828816E-212
 7.575645499390E-225
 1.635230154756E-238
 7.008506535448E-251
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  8.936000000000E-264,  8.947000000000E-264 ]
```

1.10.8.3 arctan(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arctan–Funktion:

$$\arctan(x) := \{y \,|\, y = \arctan(\zeta),\ \zeta \in x \subset (-\infty, +infty)\}$$

Im Programm wird x realisiert durch die Variable `x` vom Typ `Stagg`, und es gilt: $x_j =$ `x[j]`, wobei $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß.

Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\arctan(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.8.3.1 Der Algorithmus

Zunächst wird $\arctan(x)$ im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen $\arctan(-x) = -\arctan(x)$ kann man sich beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\arctan(x)$ eine garantierte Einschließung zu berechnen, wenn gilt

$$0 < \inf(x) \leq \sup(x) < +\infty, \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Die Einschließung von $\arctan(x)$ erfolgt mit Hilfe der Funktionalgleichungen:

$$\arctan(x) = \begin{cases} \frac{\pi}{2} - \arcsin\left(\frac{1}{\sqrt{1+x^2}}\right) & \text{falls} \quad \inf(x) > 10^{+15} \\ \arcsin\left(\frac{x}{\sqrt{1+x^2}}\right) & \text{sonst} \end{cases} \tag{1.100}$$

Für $\inf(x) > 10^{+15}$ ist die Auswertung des oberen Terms in (1.100) unkritisch, während im staggered Format die Untergrenze des Intervallarguments des unteren arcsin–Terms nicht größer werden kann als $1 - 10^{-267}$. Wegen $\arcsin(1 - \varepsilon) \approx \pi/2 - \sqrt{2\varepsilon}$ kann der untere Term daher für $\inf(x) \to \infty$ nicht beliebig genau eingeschlossen werden; vergl. dazu das obere Beispiel auf Seite 144. Beachten Sie bitte, daß die in (1.100) angegebenen Terme mit dem Argument x alles Wertebereiche sind. Demnach gilt z.B.

$$\arcsin\left(\frac{x}{\sqrt{1+x^2}}\right) := \left\{y \,|\, y = \arcsin\left(\frac{\zeta}{\sqrt{1+\zeta^2}}\right),\ \zeta \in x\right\}$$

1.10.8.3.2 Der Programm-Quelltext

```
function arctan_ge0(var t : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet wird Einschliessung von arctan(t) fuer ein }
{ schmales Intervall t, falls gilt:       inf(t) > 0;  }
{ Benutzt wird:   arctan(t) = arcsin( t/sqrt(1+t*t) )  }
{ Fuer   inf(t) > 10^15   wird benutzt:                }
{ arctan(t) = Pi/2 - arcsin(1/sqrt(1+t^2));            }
var t1 : Stagg;
   p   : integer;
begin
   p := Stagg_prec; allocate(t1,-1..p+1); set_Stagg_prec(p+1);
   t1 := change_length(t);  t1 := t1 / sqrt1px2(t1);
   t1 := arcsin(t1);  set_Stagg_prec(p);
   arctan_ge0 := change_length(t1);  free(t1);
end; { arctan_ge0 }

global function arctan(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
   t1        : Stagg;
    d        : real;
   neg,grob : boolean;
   p        : integer;
begin
   t := x;  Normal(t);  neg := false;  grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := arctan( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
     if t[1] > 1e15 then t := Pih_sm-arcsin_ge0(1/sqrt1px2(t))
                    else t := arctan_ge0(t);
   if neg then t := -t;
   arctan := t;
end; { arctan }
```

1.10.8.3.3 Numerische Ergebnisse

Mit dem Programm

```
program tests;
{ y := arctan(x) liefert Einschliessung von arctan(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x : Stagg = ?');  read(x);
       y := arctan(x);
       writeln('arctan(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man für arctan(x) die folgenden Einschließungen:

```
Stagg_prec = ?
8      <enter>
x : Stagg = ?
8 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0   <enter>
arctan(x) enthalten in:
 7.853981633974E-001
 4.830961566084E-014
 5.819875721049E-027
 2.923498437764E-040
 5.524373614807E-053
 6.954101571552E-066
 2.496570087063E-079
 3.552926699553E-092
[  7.021628320576E-105,  7.021628320577E-105 ]
```

```
Stagg_prec = ?
19     <enter>
x : Stagg = ?
19 Stagg components and 1 interval are needed in this sequence
9.999999999999e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0
                                             0 0 0 0 0 <enter>
arctan(x) enthalten in:
 1.570796326794E+000
 8.966192313216E-013
 9.163975144209E-026
 8.584699687552E-039
 9.104874722961E-052
 5.390820314310E-065
 4.499314017412E-078
 6.710585339910E-091
 7.404325664115E-104
 3.323546922304E-117
 7.752911158626E-130
 7.970406424055E-143
 8.725142051350E-156
 9.692605527798E-169
 2.231147447746E-182
 5.190982214405E-195
 4.878329667230E-208
 6.423782411689E-221
 3.391582635600E-234
[  9.545728241834E-247,  9.545728241835E-247 ]

Stagg_prec = ?
5      <enter>
x : Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
1e-30 0 0 0 0 0     <enter>
arctan(x) enthalten in:
 9.999999999999E-031
 9.999999999999E-044
 9.999999999999E-057
 9.999999999999E-070
 9.999999966666E-083
[  6.666666666666E-096,  6.666666666667E-096 ]
```

Die Funktion $f(\zeta) := \arctan(\arcsin(\zeta)) - \arcsin(\arctan(\zeta))$ besitzt für $\zeta \in I\!R$ die Potenzreihe

$$f(\zeta) = \frac{1}{30} \cdot \zeta^7 - \frac{13}{756} \cdot \zeta^9 + \frac{2329}{75600} \cdot \zeta^9 \mp \ldots \qquad |\zeta| < 1,$$

so daß bei der Auswertung von f für Argumente $|\zeta| \ll 1$ starke Auslöschung zu erwarten ist. Tatsächlich liefert das Programm

```
program beispiel;
use rrgi_ari;  { Intervallrechnung mit dem Typ rrG }
var x,y : rrGinterval;
begin
   x := 1e-30;  { Punktfoermiges Intervallargument }
   y := arctan( arcsin(x)) - arcsin(arctan(x));
   writeln('y = ',y); { Ausgabe der Einschliessung }
end.
```

für das punktförmige Intervallargument $x = 10^{-30}$ die sehr grobe Einschließung

```
y = [ {-6.084300000008E-001,  -3.042210843005E-026,    -53 },
      { 6.084300000008E-001,   3.042210843005E-026,    -53 } ]
```

die den exakten Wert $f(10^{-30})$ zwar enthält aber noch nicht einmal das richtige Vorzeichen liefert! Erst durch Anwendung der staggered Arithmetik erhält man mit dem folgenden Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    writeln('x : Stagg = ?');  read(x);
    y := arctan(arcsin(x)) - arcsin(arctan(x));
    writeln('f(x) enthalten in: ');
    write(y);
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);   main;
end.
```

eine sehr viel genauere Einschließung für $f(10^{-30})$ mit noch 54 korrekten Dezimalziffern:

```
Stagg_prec = ?
18     <enter>
x : Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e-30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   <enter>
f(x) enthalten in:
 3.333333333333E-212
 3.333333333333E-225
 3.333333333333E-238
 3.333333333333E-251
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  3.303000000000E-264,  3.364000000000E-264 ]
```

Wenn man also z.B. im einfachen `rrGinterval`–Format wegen zu starker Auslöschung viel zu grobe Einschließungen erhält, kann man durch Einsatz der staggered Arithmetik bei Wahl eines hinreichend großen Präzisionsparameters `Stagg_prec` noch ausreichende Genauigkeit erzielen.

1.10.8.4 $\mathbf{arccot}(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arccot–Funktion:

$$\operatorname{arccot}(x) := \{y \mid y = \operatorname{arccot}(\zeta),\ \zeta \in x \subset (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch die Variable `x` vom Typ `Stagg`, und es gilt: $x_j =$ `x[j]`, wobei $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß.

Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\arctan(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.8.4.1 Der Algorithmus

Die Einschließung des Wertebereichs $\operatorname{arccot}(x)$ erfolgt mit den Funktionalgleichungen:

$$\operatorname{arccot}(x) = \begin{cases} \arctan(1/x) & \text{falls } \inf(x) > 10^3 \\ \pi/2 - \arctan(x) & \text{sonst} \end{cases}$$

1.10.8.4.2 Der Programm-Quelltext

```
function arccot(var x : Stagg) : Stagg[-1..Stagg_prec];
var t : Stagg[-1..Stagg_prec];
begin
   t := x; Normal(t);
   if t[1] > 1e3 then arccot := arctan(1/t)
                 else arccot := Pih_sm - arctan(x);
end; { arccot }
```

Beachten Sie bitte, daß der Prozeduraufruf `Normal(t)` die Einschließung $x \subseteq$ `t` realisiert und daß sich die von Null verschiedenen `t[`j`]` für $j = 1, 2, \ldots$ nach dem Prozeduraufruf nicht mehr überlappen.

1.10.8.4.3 Numerische Ergebnisse

Mit dem **PASCAL-XSC** Programm

```
program tests;
{ y := arccot(x) liefert Einschliessung von arccot(x) }
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x : Stagg = ?');  read(x);
       y := arccot(x);
       writeln('arccot(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

erhält man für arccot(x) die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x : Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0 0   <enter>
arccot(x) enthalten in:
 1.570796326794E+000
 8.966192313216E-013
 9.163975144209E-026
 8.584699687552E-039
 9.104874722961E-052
 5.390820314310E-065
[  4.499314017412E-078,  4.499314017413E-078 ]
```

```
Stagg_prec = ?
18    <enter>
x : Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      <enter>
arccot(x) enthalten in:
 9.999999999999E-011
 9.999999666666E-024
 6.666666666666E-037
 6.866666666666E-050
 6.666666652380E-063
 9.523809523809E-076
 5.249206349206E-089
 3.492063491154E-102
 4.011544011544E-115
 1.162093462093E-129
 4.620934620267E-142
 9.542679542679E-155
 5.427383662089E-168
 5.444424856136E-181
 9.304929676446E-194
 7.043152433050E-207
 2.658676033284E-220
 6.885589246606E-233
[  8.802053418552E-246,  8.802053418553E-246 ]

Stagg_prec = ?
7      <enter>
x : Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
-5 0 0 0 0 0 0 0    <enter>
arccot(x) enthalten in:
 2.944197093739E+000
 9.124800925936E-013
 1.808471259074E-026
 9.584295587253E-039
 7.194572556520E-052
 6.678243630796E-065
 5.212895336997E-078
[  3.048290238695E-091,  3.048290238696E-091 ]
```

1.10.9 Die hyperbolischen Funktionen

1.10.9.1 sinh(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen sinh–Funktion:

$$\sinh(x) := \{y \,|\, y = \sinh(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung ub(x) = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von sinh(x) erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.9.1.1 Der Algorithmus

Zunächst wird der Wertebereich sinh(x) mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\sinh(-x) \equiv -\sinh(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für sinh(x) eine garantierte Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

sinh(x) ist definiert durch:

$$\sinh(x) := \frac{e^x - e^{-x}}{2} \tag{1.101}$$

Wertet man sinh(x) auf dem Rechner im Bereich $0 < \sup(x) < 0.6$ nach (1.101) aus, so erhält man um so stärkere Auslöschung, je näher x am Ursprung liegt. In diesem Bereich approximieren wir daher sinh(x) durch eine abgebrochenen Taylor–Reihe:

$$\sinh(x) \approx x \cdot P_N(x) := x \cdot \sum_{k=0}^{N} \frac{x^{2k}}{(2k+1)!}$$

In $0 < \inf(x) \leq \sup(x)$ lautet daher der Grobalgorithmus:

$$\sinh(x) := \begin{cases} \dfrac{e^x - e^{-x}}{2} & \text{falls } \inf(x) > 0.6 \\ x \cdot P_N(x) + \Delta & \text{sonst} \end{cases}$$

wobei das Intervall Δ mit Hilfe des absoluten Approximationsfehlers so zu bestimmen ist, daß eine Einschließung von $\sinh(x)$ garantiert wird.

Da nach (1.101) e^x mit Hilfe der schon implementierten `exp(x)`-Funktion eingeschlossen wird und weil die Subtraktion und Division auf dem Rechner als staggered Intervalloperationen interpretiert werden, erhält man damit für $\sinh(x)$ automatisch eine Einschließung seines Wertebereichs. Die Einschließung dieses Wertebereichs wird also schon dadurch garantiert, daß der Funktionsterm in $\inf(x) > 0.6$ intervallmäßig ausgewertet wird, [2, Seite 31].

Es bleibt damit nur noch die Berechnung von N und Δ, falls $\sinh(x)$ im Bereich $0 < \inf(x) \leq \sup(x) \leq 0.6$ durch $x \cdot P_N(x)$ zu approximieren ist. Im Falle $\inf(x) > 10^{-4}$ ist jedoch eine zusätzliche Argumentreduktion erforderlich, da sonst der Polynomgrad N viel zu groß gewählt werden muß.

Die Argumentreduktion wird realisiert mit Hilfe der Funktionalgleichung

$$\sinh(x) = \sinh\left(\frac{x}{3}\right) \cdot \left[3 + 4 \cdot \sinh^2\left(\frac{x}{3}\right)\right] \tag{1.102}$$

Ersetzt man in (1.102) das Intervall x durch $x/3$, so erhält man für $\sinh(x/3)$ einen Funktionsausdruck in $\sinh(x/3^2)$. Ersetzt man dann $\sinh(x/3)$ rechts in (1.102) durch diesen Funktionsterm, so erhält man für $\sinh(x)$ einen Ausdruck in $\sinh(x/3^2)$. Nach p Schritten erhält man dann für $\sinh(x)$ einen Term in $\sinh(x/3^p)$, und mit hinreichend großem p liegt das reduzierte Intervall-Argument $x/3^p$ beliebig dicht am Ursprung. Im Programm wird p so groß gewählt, daß $x/3^p \subset (0, 10^{-4}]$ erfüllt wird.

Ist dann t eine Einschließung von $\sinh(x/3^p)$, die durch Approximation durch ein Taylorpolynom und durch Berücksichtigung des Approximationsfehlers berechnet wurde, so erhält man für $\sinh(x)$ eine Einschließung durch folgende Schleife:

`For j := 1 to p do t := t*(3+4*(t*t));` $\quad\rightsquigarrow\quad \sinh(x) \subseteq$ `t`

Abschätzung des Approximationsfehlers:

$$\text{Für } r \in \mathbb{R} \text{ gilt:} \qquad \sinh(r) = r \cdot \sum_{k=0}^{\infty} \frac{r^{2k}}{(2k+1)!}$$

Mit dem Approximationspolynom $r \cdot P_N(r)$ vom Grade N und mit

$$P_N(r) := \sum_{k=0}^{N} \frac{r^{2k}}{(2k+1)!}$$

erhält man dann für den Betrag des absoluten Approximationsfehlers ε_a für $r > 0$ die Abschätzung:

$$\begin{aligned}
|\varepsilon_a| &= \sinh(r) - r \cdot P_N(r) = r \cdot \sum_{k=N+1}^{\infty} \frac{r^{2k}}{(2k+1)!} \\
&= \frac{r^{2N+3}}{(2N+3)!} \sum_{k=N+1}^{\infty} \frac{r2k - (2N+2) \cdot (2N+3)!}{(2k+1)!} \\
&\le \frac{r^{2N+3}}{(2N+3)!} \sum_{k=N+1}^{\infty} r2k - (2N+2) \\
&= \frac{r^{2N+3}}{(2N+3)!} \cdot \frac{1}{1-r^2} =: \alpha(r), \quad \text{falls } \ 0 \le r < 1;
\end{aligned}$$

Wir erhalten damit für $0 < r < 1$ das Ergebnis;

$$0 \le r \cdot P_N(r) \le \sinh(r) \le r \cdot P_N(r) + \alpha(r)$$

Bedeutet dann $t := x/3^p$ das Argument–Intervall, in dem $\sinh(t)$ durch den Wertebereich $t \cdot P_N(t)$ zu approximieren ist, so gilt:

$$\sinh(t) \subseteq t \cdot P_N(t) + [0, +\texttt{eps}], \ \ \texttt{eps} := \frac{t.\texttt{sup}^{2N+3}}{(2N+3)!} \cdot \frac{1}{1 - t.\texttt{sup}^2} \tag{1.103}$$

Um auf der Maschine für $\alpha(r) \le$ `eps` mit `eps` wirklich eine Oberschranke zu gewinnen, muß der Funktionsterm für $\alpha(r)$ intervallmäßig ausgewertet werden. Im Programm erfolgt dies innerhalb der Prozedur `sinh_N_e` durch die Anweisungen:

```
xi := intval(t.sup);            { xi : interval }
xi := power(xi,2*N+3) / ( intval(k)*(1-xi*xi) );
eps := xi.sup;                  { eps : real    }
```

Berechnung des Polynomgrades N:

Die Approximation von $\sinh(x/3^p)$ erfolgt für $x/3^p \subset (0, 10^{-4}]$, so daß mit $\zeta \in x/3^p$ gilt[12]:

$$\sinh(\zeta) \approx \zeta$$

Zu gegebener Präzision $S :=$ `Stagg_prec` muß im `real`–Format für den Betrag des absoluten Fehlers δ gelten:

$$|\delta| \approx 10^p, \quad \text{mit} \quad p := \text{expo}(\zeta) - 1 - 13 \cdot (S+1);$$

Da im `real`–Format 10^{-267} die kleinste darstellbare positive Zahl ist, macht es keinen Sinn, p kleiner zu wählen als -267. Im Falle

$$\text{expo}(\zeta) - 1 - 13 \cdot (S+1) < -267$$

wird daher $p = -267$ gesetzt. Der Polynomgrad N ergibt sich dann aus der vereinfachten Forderung:

$$\frac{\zeta^{2N+3}}{(2N+3)!} \cdot \frac{1}{1-\zeta^2} \approx \frac{\zeta^{2N+3}}{(2N+3)!} < 10^p \qquad \Longleftrightarrow$$

$$(2N+3) \cdot \log_{10}(\zeta) - \log_{10}(2N+3)! < p \tag{1.104}$$

Im Programm wird innerhalb der Prozedur `sinh_N_e` der Polynomgrad N schrittweise um 1 erhöht, bis (1.104) erfüllt ist. Beachten Sie bitte, daß die auftretenden Ausdrücke zur Berechnung von N so weit vereinfacht werden können, das eine wesentliche Verfälschung des Polynomgrades gerade noch nicht eintritt. Aus Laufzeitgründen wird man außerdem die Berechnung von N im einfachen `real`–Format vornehmen.

Polynomauswertung:

Mit dem möglicherweise reduzierten Intervallargument $x \subset (0, 10^{-4}]$ gilt:

$$x \cdot P_N(x) = x \cdot \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k+1)!}, \quad \text{und} \quad \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k+1)!} =$$

$$= \left(\cdots \left(\left(\frac{x^2}{(2N+1) \cdot 2N} + 1 \right) \cdot \frac{x^2}{(2N-1)(2N-2)} + 1 \right) \cdots + 1 \right) \cdot \frac{x^2}{3 \cdot 2} + 1$$

Auf der Maschine wird die Auswertung der letzten Summe daher realisiert durch die einfache Schleife:

[12] Im Programm ist $\zeta = \sup(x/3^p)$

```
         y := x*x;  P := 1;
         For j := N downto 1 do
         begin
            k := 2*j;  pot := k*(k+1);
            P := P * y / pot + 1;
         end;
         P := x * P;   { P: Approximation fuer sinh(x) }
```

Nach (1.103) liefert dann die Anweisung

```
P := P + Staggintval(0,eps);
```

mit `P` eine garantierte Einschließing von $\sinh(x) \subseteq$ `P`, falls $\inf(x) > 0$.

1.10.9.1.2 Der Programm-Quelltext

```
procedure sinh_N_e(   S : integer; t : real;
                  var N : integer; var eps : real);
{ Berechnet Polynomgrad N und Oberschranke }
{ eps des absoluten Approximationsfehlers. }
{ t muss x.sup sein!!!                     }
var lgt,k : real;
   p      : integer;
   xi     : interval;
begin
   lgt := log10(t);
   p := expo(t) - 1 - 13*(S+1);
   if p < -267 then p := -267;
   N := 0;  k := 3;
   repeat
      N := N+1;
      k := k *< (2*N+2)*(2*N+3);
   until (2*N+3)*lgt - log10(k) < p;
   xi := intval(t);
   xi := power(xi,2*N+3) / ( intval(k)*(1-xi*xi) );
   eps := xi.sup;
end;

function sinh_ge0(var x : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet sinh(x), falls x > 0;     diese Bedingung wird }
{ in dieser Funktion jedoch nicht abgefragt!               }
```

```
{ Blomquist, 11.03.2001;                                  }
const c = 1e-4;
   rlg3 = 2.096; { rlg3 > 1/log_10(3) }
var y,t,x1     : Stagg[-1..Stagg_prec];
   t1          : Stagg;
   eps         : real;
   xi          : interval;
   p,j,N,k,pot : integer;
   reduce      : boolean;
begin
   if x[1] > 0.6 then
   begin { Anwendung der Funktionalgleichung: }
      j := get_Stagg_prec;  allocate(t1,-1..j+1);
      set_Stagg_prec(j+1);
      t1 := exp( change_length(x) );
      t1 := 0.5 * (t1 - 1/t1);
      set_Stagg_prec(j);
      t := change_length(t1);  free(t1);
   end           else
   begin
      reduce := x[1] > c;
      x1 := x;
      if reduce then
      begin { Argument-Reduktion ist notwendig: }
         eps := rlg3 * (4 + log10(x[1]));
         p := trunc(eps) + 1;
         j := round( power(3,p) );        { j = 3^p }
         x1 := x1 / j;  { Argument-Reduktion fertig }
      end;
      { Berechnung des Polynomgrades: }
      xi := ival(x1);  j := get_Stagg_prec;
      sinh_N_e(j,xi.sup,N,eps);
      { Polynomauswertung: }
      y := x1*x1;  t := 1;
      For j := N downto 1 do
      begin
         k := 2*j;  pot := k*(k+1);
         t := t*y/pot + 1;
      end;
      t := x1 * t;         { t = Approximation fuer sinh(x1) }
```

```
      t := t + Staggintval(0,eps);          { Approx-Fehler! }
      { sinh(x1) ist jetzt enthalten in t }
      if reduce then   { Ruecktransformation: }
         For j := 1 to p do t := t*(3+4*(t*t));
   end;
   sinh_ge0 := t;
end; { sinh_ge0 }

function sinh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
    d        : real;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := sinh( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      t := sinh_ge0(t);
   if neg then t := -t;
   sinh := t;
end; { sinh }
```

1.10.9.1.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := sinh(x);
       writeln('sinh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich sinh(x) die folgenden Einschließungen:

```
Stagg_prec = ?
8      <enter>
x: Stagg = ?
8 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0 0 0    <enter>
sinh(x) enthalten in:
 1.000000000000E-020
 1.666666666666E-061
 6.666666666666E-074
 6.666666666666E-087
 6.675000000000E-100
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  3.968253968256E-144 ]
```

```
Stagg_prec = ?
22     <enter>
x: Stagg = ?
22 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      <enter>
sinh(x) enthalten in:
 1.000000000000E-020
 1.666666666666E-061
 6.666666666666E-074
 6.666666666666E-087
 6.675000000000E-100
 1.984126984126E-144
 9.841269841269E-157
 8.412698412698E-170
 4.129739858906E-183
 5.255731922398E-196
 5.890652557319E-209
 2.239861411736E-222
 4.117364117364E-235
 1.173641173641E-248
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  1.736411000000E-261,  1.736413000000E-261 ]

Stagg_prec = ?
2      <enter>
x: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [0,1e-267]     <enter>
sinh(x) enthalten in:
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  2.000000000000E-267 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
0.7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       <enter>
sinh(x) enthalten in:
 7.585837018395E-001
 3.350345987464E-014
 7.592768154154E-027
 9.376142170255E-040
 2.475713890193E-053
 1.268865392982E-066
 3.037286982162E-079
 6.163533655163E-092
 5.634573356386E-105
 4.311849176219E-119
 7.567176139192E-132
 3.099469910280E-145
 5.396853378098E-158
 9.619419464743E-171
 7.032618784482E-184
 4.043780256570E-197
 6.194216196342E-210
 6.708652262791E-223
[  5.165564061031E-236,  5.165564061032E-236 ]

Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
588 0 0 0 0 0 0 0      <enter>
sinh(x) enthalten in:
 1.159111897116E+255
 5.927866480688E+242
 6.756977834700E+229
 1.416665313252E+215
 4.497332498653E+202
 9.659802414741E+189
 2.659064256276E+176
[  5.380801551108E+163,  5.380801551109E+163 ]
```

1.10.9.2 cosh(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen cosh–Funktion:

$$\cosh(x) := \{y \,|\, y = \cosh(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\cosh(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.9.2.1 Der Algorithmus

Die Funktion $\cosh(x)$ ist definiert durch:

$$\cosh(x) := \frac{e^x + e^{-x}}{2} \tag{1.105}$$

Da $\cosh(x)$ eine gerade Funktion ist, gilt für den Wertebereich die Identität $\cosh(x) \equiv \cosh(|x|)$, so daß man sich auf Intervallargumente x beschränken kann mit $\inf(x) \geq 0$. Im Falle $x = 0$ wird der Funktionswert auf 1 gesetzt, so daß danach die Bedingung $\inf(x) > 0$ erfüllt ist.

$\cosh(x)$ wird mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn der relative Durchmesser von x größer ist als 10^{-14} und wenn $\sup(x) > 10^{-13}$ erfüllt ist. In allen anderen Fällen wird $\cosh(x)$ mit Hilfe von (1.105) eingeschlossen, wobei die Rechnung mit um 1 erhöhter Präzision erfolgt.

Liegt das Argument x zu dicht am Ursprung, so kan es vorkommen, daß auf der Maschine durch interne Intervallüberschätzungen die Unterschranke der Einschließung von $\cosh(x)$ kleiner als 1 ausfällt. In diesem Fall wird diese Unterschranke auf 1 gesetzt, da für $r \in \mathbb{R}$ gilt: $\cosh(r) \geq 1$.

Wird auf der Maschine der Definitionsbereich der Exponentialfunktion in (1.105) überschritten, so erfolgt eine entsprechende Fehlermeldung.

Der Wertebereich der Funktion $\cosh(x)$ kann mit der maximalen Präzision S = `Stagg_prec` $= 18$ eingeschlossen werden.

1.10.9.2.2 Der Programm-Quelltext

```
function cosh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14; { Rel. Durchmesser > c  --> grobe Einschlg. }
   c1   = 1e-13;
var t      : Stagg[-1..Stagg_prec];
   t1      : Stagg;
   S       : integer;
   test_D : boolean;
begin
   t := x;
   t := abs(t);
   Normal(t);
   if t = 0 then t := 1 else
     if ((t[0]->t[-1]) /< (t[1]+t[0])>c) and (t[1]+t[0] > c1)
        then t := cosh( ival(t) ) else
        begin  { Jetzt ein schmales Intervall-Argument: }
           test_D := expo(t[1]) < -10;
           S := get_Stagg_prec; { Alte Praezision festhalten }
           allocate(t1,-1..S+1);  { Speicher fuer t1 anlegen }
           set_Stagg_prec(S+1);   { Praezision um 1 erhoehen }
           t1 := change_length(t); { t1=t in erhoehter Praez.}
           t1 := exp(t1);
           t1 := 0.5 * (t1 + 1/t1);
           set_Stagg_prec(S);  { Alte Praezision realisieren }
           t := change_length(t1);{ t1 in alte Praez. runden }
           free(t1);       { Speicherplatz von t1 freigeben! }
           if test_D then  { Es sollte gelten:  cosh(x) >= 1 }
             if inf(t) < 1 then
                                t := hull(Staggintval(1),sup(t));
        end;
   cosh := t;
end; { cosh }
```

Der Aufruf `y := cosh(x)` liefert mit `y` eine garantierte Einschließung des Wertebereichs cosh(`x`).

1.10.9.2.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := cosh(x);
       writeln('cosh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);  main;
end.
```

liefert für den Wertebereich $\cosh(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
2      <enter>
x: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [0,1e-200]     <enter>
cosh(x) enthalten in:
 1.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  5.000000000025E-201 ]

Stagg_prec = ?
1      <enter>
x: Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 [0,1e-5]  <enter>
cosh(x) enthalten in:
 0.000000000000E+000
[  1.000000000000E+000,  1.000000000051E+000 ]
```

```
Stagg_prec = ?
4      <enter>
x: Stagg = ?
4 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0   <enter>
cosh(x) enthalten in:
 1.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]

Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
66.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0      <enter>
cosh(x) enthalten in:
 3.797983301053E+028
 6.668167317236E+015
 6.380491557971E+002
 3.452300690819E-011
 2.139822441707E-024
 1.876422455190E-037
 3.146156478953E-050
 7.346074558667E-063
 8.299589646381E-076
 1.807035468840E-089
 3.017537956569E-102
 8.543065611280E-115
 7.723185322446E-129
 2.248461592244E-142
 9.458511046477E-155
 7.873928478422E-168
 1.044497310434E-181
 9.925145001146E-194
[  2.821714476006E-207,  2.821714476007E-207 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
cosh(x) enthalten in:
 1.000000000000E+000
 5.000000000000E-041
 4.166666666666E-082
 6.666666666666E-095
 6.666666666666E-108
 6.680555555555E-121
 5.555555555555E-134
 5.555555555555E-147
 5.555580357142E-160
 8.571428571428E-173
 5.714285714285E-186
 7.142857170414E-199
 4.620811287477E-212
 9.541446208112E-225
 8.747795414482E-238
 9.578857356635E-251
 0.000000000000E+000
 0.000000000000E+000
[  1.300000000000E-264,  1.390000000000E-264 ]

Stagg_prec = ?
5      <enter>
x: Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
1e-80 0 0 0 0 0    <enter>
cosh(x) enthalten in:
 1.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  4.999999999861E-161,  5.000000000145E-161 ]
```

1.10.9.3 $\tanh(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen tanh–Funktion:

$$\tanh(x) := \{y \,|\, y = \tanh(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\tanh(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.9.3.1 Der Algorithmus

Der Wertebereich $\tanh(x)$ läßt sich darstellen als:

$$\tanh(x) := \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.106}$$

$$= 1 - \frac{2}{e^{2x} + 1} \tag{1.107}$$

$$= 1 - \frac{2 \cdot e^{-2x}}{1 + e^{-2x}} \tag{1.108}$$

$$= \frac{\sinh(x)}{\cosh(x)} \tag{1.109}$$

Zunächst wird der Wertebereich $\tanh(x)$ mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\tanh(-x) \equiv -\tanh(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\tanh(x)$ eine garantierte Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Der Wertebereich $\tanh(x)$ ist also einzuschließen für relativ schmale Intervall-Argumente x, die rechts vom Ursprung liegen. Der Grobalgorithmus lautet daher:

$$\tanh(x) = \begin{cases} 1 - \dfrac{2e^{-2x}}{1+e^{-2x}} & \inf(x) > 293 \\ \dfrac{\sinh(x)}{\cosh(x)} & \sup(x) < 0.6 \\ 1 - \dfrac{2}{e^{2x}+1} & \text{sonst mit höhere Präzision} \end{cases}$$

Wir betrachten zunächst den Fall: $\inf(x) > 293$

Da $\tanh(\zeta)$ monoton wächst, gilt mit $\alpha := 2/(e^{2\cdot\inf(x)} + 1)$ für $\zeta \in x$ die Ungleichung:

$$1 - \alpha \leq \tanh(\zeta) < 1 \quad \rightsquigarrow \quad \tanh(x) \subset 1 - [0, \alpha]$$

Um für α eine Oberschranke zu berechnen, wird α auf der Maschine intervallmäßig ausgewertet, indem man $\inf(x)$ durch das Intervall x ersetzt. Aus Laufzeitgründen erfolgt die Intervallauswertung im einfachen `interval`–Format. Zur Vermeidung von Overflow bei zu großen Argumenten in $(e^{2\cdot\inf(x)} + 1)$ benutzt man die Identität

$$\alpha(x) := \frac{2}{e^{2x}+1} \equiv \frac{2e^{-2x}}{1+e^{-2x}},$$

wobei e^{-2x} unter der Voraussetzung $\inf(x) > 0$ im einfachen `real`–Format für jedes Interval $-2x$ eingeschlossen wird, das auf der Maschine darstellbar ist, d.h. also für alle x mit $2 \cdot \sup(x) < 10^{+256}$. Im Programm erfolgt die Auswertung durch:

```
if t[1] > 293 then
begin
   xi := ival(t); { xi: interval ist Einschliessung }
   xi := exp(-2*xi);   { des Argument-Intervalls x. }
   xi := 2*xi / (1+xi);
    t := 1 - Staggintval(0,xi.sup);
end   { Der Wertebereich    tanh(x)    ist jetzt im }
      { staggered Intervall t garantiert enthalten! }
```

Im Fall $\sup(x) < 0.6$ wird $\tanh(x)$ eingeschlossen mit Hilfe der bereits implementierten Funktionen `sinh(x)` und `cosh(x)`. In allen anderen Fällen benutzt man die Funktionalgleichung:

$$\tanh(x) = 1 - \frac{2}{e^{2x}+1},$$

wobei die rechte Seite mit Hilfe der bereits implementierten Exponentialfunktion mit der um 1 erhöhten Präzision ausgewertet wird, um die unvermeidbaren Intervallüberschätzungen bei der Einschließung von $e^{2\cdot x}$ in der einfachen Präzision $S =$ `Stagg_prec` zu unterdrücken.

1.10.9.3.2 Der Programm-Quelltext

```
function tanh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
   t1        : Stagg;
    d        : real;
   S         : integer;
   xi        : interval;
   neg,grob : boolean;
begin
   if Stagg_prec > 18 then s_fu1_math_error(15);
   t := x;  Normal(t);   neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := tanh( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      if t[1] > 0.6 then
         { Anwendung der Funktionalgleichung: }
         if t[1] > 293 then
         begin
            xi := ival(t);
            xi := exp(-2*xi);
            xi := 2*xi / (1+xi);
            t := 1 - Staggintval(0,xi.sup);
         end          else
         begin
            S := get_Stagg_prec;  allocate(t1,-1..S+1);
            set_Stagg_prec(S+1);
```

```
            t1 := exp( 2*change_length(t) ) + 1;
            t1 := 1 - 2/t1;
            set_Stagg_prec(S);
            t := change_length(t1);  free(t1);
         end        else t := sinh(t) / cosh(t);
   if neg then t := -t;
   tanh := t;
end; { tanh }
```

1.10.9.3.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := tanh(x);
       writeln('tanh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }

begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);  main;
end.
```

liefert für den Wertebereich $\tanh(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
1       <enter>
x: Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 [0,0]    <enter>
tanh(x) enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
tanh(x) enthalten in:
 9.999999999999E-001
 9.999999999999E-014
 9.999999999999E-027
 9.999999999999E-040
 9.999999999999E-053
 9.999999999999E-066
 9.999999999999E-079
 9.999999999999E-092
 9.999999999999E-105
 9.999999999999E-118
 9.999999999999E-131
 9.999999999999E-144
 9.999999999999E-157
 9.999999999999E-170
 9.999999999999E-183
 9.999999999999E-196
 9.999999999999E-209
 9.999999999999E-222
[  9.999999999999E-235,  1.000000000000E-234 ]

Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
1e-10 0 0 0 0 0 0 0    <enter>
tanh(x) enthalten in:
 9.999999999999E-011
 9.999999666666E-024
 6.666666666666E-037
 6.799999999999E-050
 9.999999994603E-063
 1.746031746031E-076
 7.462504409171E-089
[  7.583774249554E-103,  7.583774249555E-103 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
tanh(x) enthalten in:
 9.640275800758E-001
 1.688394641372E-014
 4.100923150255E-027
 2.997624093477E-041
 6.048263217413E-054
 1.079463176102E-067
 2.559474850045E-081
 2.076891494616E-094
 1.221892459666E-107
 8.273067021913E-120
 2.033454748595E-134
 6.472914624336E-147
 7.515945063168E-160
 1.200724595065E-173
 8.562057196950E-186
 5.507398350541E-199
 3.184843572307E-212
 1.198131843339E-225
[  6.258626382972E-239,  6.258626382973E-239 ]

Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
1e-200 0 0 0 0 0 0 0   <enter>
tanh(x) enthalten in:
 9.999999999999E-201
 9.999999999999E-214
 9.999999999999E-227
 9.999999999999E-240
 9.999999999999E-253
 0.000000000000E+000
 0.000000000000E+000
[  9.900000000000E-266,  1.020000000000E-265 ]
```

1.10.9.4 $\coth(x)$

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen coth–Funktion:

$$\coth(x) := \{y \mid y = \coth(\zeta),\ \zeta \in x;\ x \subset (-\infty, 0) \vee x \subset (0, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: $x_j =$ `x[j]`, wobei die Bedingung $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\coth(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. Der Parameter $S =$ `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.9.4.1 Der Algorithmus

Der Wertebereich $\coth(x)$ läßt sich darstellen als:

$$\coth(x) := \frac{e^x + e^{-x}}{e^x - e^{-x}} \tag{1.110}$$

$$= 1 + \frac{2}{e^{2x} - 1} \tag{1.111}$$

$$= 1 + \frac{2 \cdot e^{-2x}}{1 - e^{-2x}} \tag{1.112}$$

$$= \frac{\cosh(x)}{\sinh(x)} \tag{1.113}$$

Zunächst wird der Wertebereich $\coth(x)$ mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\coth(-x) \equiv -\coth(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\coth(x)$ eine garantierte Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Der Wertebereich $\coth(x)$ ist also einzuschließen für relativ schmale Intervall-Argumente x, die rechts vom Ursprung liegen.

Da die Auswertung des Quotienten $\cosh(x)/\sinh(x)$ auch für Punktintervall-Argumente x, die dicht am Ursprung liegen, zu grobe Einschließungen liefert, benutzt man für $\sup(x) < 10^{-13}$ die für $0 < |x| < \pi$ gültige Darstellung

$$\coth(x) = \frac{1}{x} + \frac{x}{3} - \frac{x^3}{45} \pm \ldots = \frac{1}{x} + \sum_{k=1}^{\infty} \frac{2^{2k} \cdot B_{2k}}{(2k)!} \cdot x^{2k-1}, \tag{1.114}$$

wobei die B_{2k} die Bernoullischen Zahlen sind. Der Grobalgorithmus lautet daher:

$$\coth(x) = \begin{cases} 1 + \dfrac{2e^{-2x}}{1 - e^{-2x}} & \inf(x) > 293 \\ \dfrac{\cosh(x)}{\sinh(x)} & x \subset [10^{-13}, 0.6] \\ \dfrac{1}{x} + \displaystyle\sum_{k=1}^{\infty} \frac{2^{2k} \cdot B_{2k}}{(2k)!} \cdot x^{2k-1} & x \subset (0, 10^{-13}) \\ 1 + \dfrac{2}{e^{2x} - 1} & \text{sonst mit höhere Präzision} \end{cases}$$

Wir betrachten zunächst den Fall: $\inf(x) > 293$

Da $\coth(\zeta)$ monoton fällt, gilt mit $\alpha := 2/(e^{2\cdot\inf(x)} - 1)$ für $\zeta \in x$ die Ungleichung:

$$1 < \coth(\zeta) \leq 1 + \alpha \quad \rightsquigarrow \quad \coth(x) \subset 1 + [0, \alpha]$$

Um für α eine Oberschranke zu berechnen, wird α auf der Maschine intervallmäßig ausgewertet, indem man $\inf(x)$ durch das Intervall x ersetzt. Aus Laufzeitgründen erfolgt die Intervallauswertung im einfachen `interval`–Format. Zur Vermeidung von Overflow bei zu großen Argumenten in $(e^{2\cdot\inf(x)} - 1)$ benutzt man die Identität

$$\alpha(x) := \frac{2}{e^{2x} - 1} \equiv \frac{2e^{-2x}}{1 - e^{-2x}},$$

wobei e^{-2x} unter der Voraussetzung $\inf(x) > 0$ im einfachen `real`–Format für jedes Interval $-2x$ eingeschlossen wird, das auf der Maschine darstellbar ist, d.h. also für alle x mit $2 \cdot \sup(x) < 10^{+256}$. Im Programm erfolgt die Auswertung durch:

```
if t[1] > 293 then
begin
   xi := ival(t); { xi: interval ist Einschliessung }
   xi := exp(-2*xi);   { des Argument-Intervalls x. }
   xi := 2*xi / (1-xi);
   t  := 1 + Staggintval(0,xi.sup);
end   { Der Wertebereich    tanh(x)    ist jetzt im }
      { staggered Intervall t garantiert enthalten! }
```

Im Fall $x \subset [10^{-13}, 0.6]$ wird $\cot(x)$ eingeschlossen mit Hilfe der bereits implementierten Funktionen `sinh(x)` und `cosh(x)`. Im Falle $x \subset [0.6, 293]$ benutzt man die Funktionalgleichung:

$$\coth(x) = 1 + \frac{2}{e^{2x} - 1},$$

wobei die rechte Seite mit Hilfe der bereits implementierten Exponentialfunktion mit der um 1 erhöhten Präzision ausgewertet wird, um die unvermeidbaren Intervallüberschätzungen bei der Einschließung von $e^{2\cdot x}$ in der einfachen Präzision $S =$ `Stagg_prec` zu unterdrücken.

Schließlich bleibt noch der Fall $x \subset (0, 10^{-13}]$ mit der Darstellung

$$\coth(x) = \frac{1}{x} + \sum_{k=1}^{\infty} \frac{4^k \cdot B_{2k}}{(2k)!} \cdot x^{2k-1}, \ \sup(x) < \pi; \quad a_k := 12 \cdot \frac{4^k \cdot B_{2k+2}}{(2k+2)!}$$

$$\rightsquigarrow \qquad \coth(x) = \frac{1}{x} + \frac{x}{3} \sum_{k=0}^{\infty} a_k \cdot x^{2k} = \frac{1}{x} + \frac{x}{3} \cdot P_\infty(x^2)$$

Ein Nachteil der Reihe $P_\infty(x^2)$ besteht allerdings darin, daß es zur Rechnerauswertung wegen der Bernoullischen Zahlen B_{2k} keine einfache Rekursionsformel für die Polynomkoeffizienten gibt. Zur Polynomauswertung werden daher Zähler und Nenner der rationalen Koeffizienten a_k bis zum Grade $N = 7$ im `real`–Format gespeichert und bei der ersten Übersetzung des **PASCAL-XSC** Moduls `s_fu1` in der durch $S =$ `Stagg_prec` vorgegebenen Präzision gespeichert. Bevor wir den Approximationsfehler abschätzen, der bei der Approximation $P_\infty(\zeta^2) \approx P_N(\zeta^2)$ zu berücksichtigen ist, soll zunächst nachgewiesen werden, daß $\zeta/3 \cdot P_\infty(\zeta^2)$ eine Leibniz–Reihe ist. Das Alternieren der Reihenglieder ist dabei gesichert durch das bekannte Alternieren der B_{2k}. Es bleibt daher zu zeigen, daß die Beträge der Reihenglieder eine monotone Nullfolge bilden. Mit

$$b_k(\zeta) := \frac{4^k \cdot |B_{2k}| \cdot \zeta^{2k-1}}{(2k)!}, \quad k = 1, 2, \ldots, \quad 0 < \zeta \leq 10^{-13}$$

ist daher zunächst für alle $k \in \mathbb{N}$ zu zeigen: $b_{k+1}(\zeta)/b_k(\zeta) < 1$. Zum Beweis benutzen wir die nach [1, Seite 805] gültige Ungleichung

$$\frac{2(2k)!}{(2\pi)^{2k}}\left(\frac{1}{1-2^{1-2k}}\right) > |B_{2k}| > \frac{2(2k)!}{(2\pi)^{2k}}, \qquad k = 1, 2, \ldots$$

und nach einfachen Umrechnungen erhält man daraus

$$\frac{b_{k+1}(\zeta)}{b_k(\zeta)} < \left(\frac{\zeta}{\pi}\right)^2 \cdot \frac{1}{1-2^{-2k-1}} \leq \left(\frac{\zeta}{\pi}\right)^2 \cdot \frac{8}{7}$$

Unter der Voraussetzung $\zeta < \pi \cdot \sqrt{7/8}$ gilt daher für alle k die behauptete Ungleichung: $b_{k+1}(\zeta)/b_k(\zeta) < 1$. Zum Nachweis der Leibniz–Reihe fehlt jetzt noch der Nachweis

$$\lim_{k\to\infty} b_k(\zeta) = 0$$

Daß die $b_k(\zeta)$ eine Nullfolge bilden ist jedoch automatisch gesichert, da die angegebene Reihe der coth-Funktion für $|\zeta| < \pi$ konvergiert. ■

Gilt nun für die reelle Funktion $f(\zeta)$ die Entwicklung

$$f(\zeta) = \sum_{k=0}^{\infty} b_k(\zeta) = P_\infty(\zeta)$$

und ist obige Reihe eine Leibniz-Reihe und approximiert man $f(\zeta)$ durch

$$f(\zeta) \approx P_N(\zeta) := \sum_{k=0}^{N} b_k(\zeta),$$

so gilt nach [3, Seite 259] für den Fehler $F_N(\zeta) := f(\zeta) - P_N(\zeta)$:

F_N hat dasselbe Vorzeichen wie b_{N+1} und es gilt:
$|F_N(\zeta)| < |b_{N+1}(\zeta)|$.

Danach lassen sich für $f(\zeta)$ die folgenden Ungleichungen angeben:

$$\sum_{k=0}^{N} b_k(\zeta) < f(\zeta) < \sum_{k=0}^{N} b_k(\zeta) + b_{N+1}(\zeta), \quad \text{falls } b_{N+1}(\zeta) > 0 \tag{1.115}$$

$$\sum_{k=0}^{N} b_k(\zeta) + b_{N+1}(\zeta) < f(\zeta) < \sum_{k=0}^{N} b_k(\zeta), \quad \text{falls } b_{N+1}(\zeta) < 0 \tag{1.116}$$

Bezeichnen wir jetzt mit $P_N(x), f(x)$ und $b_{N+1}(x)$ die Wertebereiche der Funktionen $P_N(\zeta), f(\zeta), b_{N+1}(\zeta)$ für $\zeta \in x$, so erhält man:

$$f(x) \subset P_N(x) + b_{N+1}(x). \qquad \text{Dabei gilt:} \tag{1.117}$$

$$b_{N+1}(x) = \begin{cases} [0, \max\limits_{\zeta \in x}\{b_{N+1}(\zeta)\}] & \text{falls } \; b_{N+1}(\zeta) > 0 \\ [\min\limits_{\zeta \in x}\{b_{N+1}(\zeta)\}, 0] & \text{falls } \; b_{N+1}(\zeta) < 0 \end{cases} \tag{1.118}$$

Beachten Sie bitte, daß die Addition rechts in (1.117) die Addition zweier Wertebereiche, d.h. also die Addition zweier Intervalle bedeutet. Wenn die Funktion $b_{N+1}(\zeta)$ mit Hilfe der Potenz ζ^{N+1} definiert ist, so lassen sich die obigen Maxima oder Minima an den entsprechenden Intervallrändern von x sehr einfach berechnen. Wenn also $f(\zeta)$ durch eine abgebrochene Leibniz–Reihe approximiert wird, so kann eine Einschließung des Wertebereichs $f(x)$ mit Hilfe des ersten weggelassenen Summanden bequem angegeben werden!

Einschließung von coth(x), $x \subset (0, 10^{-13}]$:

Mit den Bezeichnungen

$$a_k := 12 \cdot \frac{4^k \cdot B_{2k+2}}{(2k+2)!}, \quad P_\infty(x^2) := \sum_{k=0}^{\infty} a_k \cdot x^{2k}, \quad P_N(x^2) := \sum_{k=0}^{N} a_k \cdot x^{2k}$$

folgt mit den Ergebnissen von Seite 180 und mit (1.117)

$$\begin{aligned} \coth(x) &= \frac{1}{x} + \frac{x}{3} \cdot P_\infty(x^2) \\ &\subset \frac{1}{x} + \frac{x}{3} \cdot P_N(x^2) + \frac{x}{3} \cdot a_{N+1} \cdot x^{2N+2} \end{aligned} \tag{1.119}$$

Nach (1.118) gilt dabei

$$\frac{x}{3} \cdot a_{N+1} \cdot x^{2N+2} = \begin{cases} [0, \frac{a_{N+1}}{3} \cdot \sup^{2N+3}(x)] & \text{falls } \; a_{N+1} > 0 \\ [\frac{a_{N+1}}{3} \cdot \sup^{2N+3}(x), 0] & \text{falls } \; a_{N+1} < 0 \end{cases}$$

Im Programm werden Zähler und Nenner der a_k bis zum maximalen Index $k = 8$ gespeichert, so daß die Einschließung von $\coth(x)$ mit Hilfe von $P_N(x)$ bis zum maximalen Polynomgrad $N = 7$ erfolgen kann. Die Einschließung nach (1.119) wird dann für $x \subset (0, 10^{-13}]$ mit ausreichender Genauigkeit berechnet.

Berechnung des Polynomgrades $N \leq 7$

Für $\zeta \in x \subset (0, 10^{-13}]$ gilt in guter Näherung:

$$\coth(\zeta) \approx \frac{1}{\zeta} + \frac{\zeta}{3}$$

Zu gegebener Präzision $S =$ `Stagg_prec` verlangen wir im `real`–Format für den Betrag des absoluten Fehlers δ:

$$|\delta| \approx 10^p, \quad \text{mit} \quad p := \text{expo}(\zeta/3) - 1 - 13 \cdot (S+1)$$

Da im `real`–Format 10^{-267} die kleinste darstellbare positive Zahl ist, macht es keinen Sinn, p kleiner zu wählen als -267. Im Falle

$$\text{expo}(\zeta/3) - 1 - 13 \cdot (S+1) < -267$$

wird daher $p = -267$ gesetzt. Der Polynomgrad N ergibt sich dann aus der Forderung:

$$\frac{|a_{N+1}|}{3} \cdot \zeta^{2N+3} < 10^p, \quad N = 0, 1, \ldots, 7 \quad \Longleftrightarrow$$

$$\log_{10}(|a_{N+1}|) - \log_{10}(3) + (2N+3) \cdot \log_{10}(\zeta) < p \tag{1.120}$$

Beginnend mit $N = 0$ wird N schrittweise um 1 erhöht, bis die Ungleichung (1.120) erfüllt ist oder bis der Maximalwert $N = 7$ erreicht ist. Eine Einschließung des Wertebereichs $P_N(x^2)$ ergibt sich dann durch Auswertung des Polynoms nach dem Hornerschema.

Einschließung von $a_{N+1} \cdot x^{2N+3}/3$

Die Einschließung dieses Wertebereichs erfolgt durch Intervallauswertung im einfachen `real`–Format an der Obergrenze $\texttt{r} \geq \zeta \in x$ durch die folgenden Anweisungen:

```
xi := ival(x);  { xi : interval  ist Einschliessung von x }
r := xi.sup;    { es gilt: r >= zeta fuer alle zeta aus x }
xi := intval(r);       { xi : interval ist Punktintervall }
xi := power(xi,2*N+3) * intval(abs(Za[N+1]))/(Ne[N+1]*<3);
x2 := Staggintval(0.0,xi.sup);
if SIGN(Za[N+1]) > 0 then coth_S := t + x2
                     else coth_S := t - x2;
```

1.10.9.4.2 Der Programm-Quelltext

```
function coth_S(var x : Stagg) : Stagg[-1..Stagg_prec];
{ Reihe fuer   coth(x) = 1/x +(1/3)*x -+...;   x > 0 }
const lg3 = 0.4771212;  { log10(3) }
var Za,Ne : array[0..8] of real;
   t,x2   : Stagg[-1..Stagg_prec];
   xi     : interval;
   j,N,p,S  : integer;
   r,lgr,xr  : real;
begin
   Za[0] := 1;        Ne[0] := 1;
   Za[1] := -1;       Ne[1] := 15;
   Za[2] := 2;        Ne[2] := 315;
   Za[3] := -1;       Ne[3] := 1575;
   Za[4] := 2;        Ne[4] := 31185;
   Za[5] := -1382;    Ne[5] := 212837625;
   Za[6] := 4;        Ne[6] := 6081075;
   Za[7] := -3617;    Ne[7] := 5.4273594375e10;
   Za[8] := 8773;     Ne[8] := 1.299309849337e13;
                      { Ne[8] ist abgerundet! }
   { Berechnung des Polynomgrades: }
   xi := ival(x);  r := xi.sup;  lgr := log10(r);
   S := get_Stagg_prec;
   p := expo(lgr/3) -1 -13*(S+1);
   if p < -267 then p := -267;
   N := -1;
   repeat
      N := N+1;
      xr := log10(abs(Za[N+1])/Ne[N+1]) -lg3 + (2*N+3)*lgr
   until (xr < p) or (N = 7);  { N ist maximal 7 }
   { Polynomauswertung: }
   x2 := x*x;
   t := Staggintval(Za[N]) / Ne[N];
   For j := N-1 downto 0 do
      t := t * x2 + Staggintval(Za[j]) / Ne[j];
   t := 1/x + x * t / 3;
   xi := intval(r);
   xi := power(xi,2*N+3) * intval(abs(Za[N+1]))/(Ne[N+1]*<3);
   x2 := Staggintval(0.0,xi.sup);
```

```
   if SIGN(Za[N+1]) > 0 then coth_S := t + x2
                        else coth_S := t - x2;
end; { coth_S }

global
function coth(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
var t        : Stagg[-1..Stagg_prec];
   t1        : Stagg;
    d        : real;
   S         : integer;
   xi        : interval;
   neg,grob : boolean;
begin
   t := x;  Normal(t);   neg := false;
   grob := 0 in t;
   if not grob then
   begin
      if Stagg_prec > 18 then s_fu1_math_error(15);
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := coth( ival(t) )
   else  { t ist schmal, und es gilt: inf(t) > 0 }
      if t[1] > 0.6 then
         { Anwendung der Funktionalgleichung: }
         if t[1] > 293 then
         begin
            xi := ival(t);
            xi := exp(-2*xi);
            xi := 2*xi / (1-xi);
            t := 1 + Staggintval(0,xi.sup);
         end          else
         begin
            S := get_Stagg_prec;  allocate(t1,-1..S+1);
            set_Stagg_prec(S+1);
            t1 := exp( 2*change_length(t) ) - 1;
            t1 := 1 + 2/t1;
```

```
            set_Stagg_prec(S);
            t := change_length(t1);  free(t1);
         end        else
         if t[1]+t[0] < 1e-13 then t := coth_S(t)  { Reihe }
                              else t := cosh(t) / sinh(t);
   if neg then t := -t;
   coth := t;
end; { coth }
```

1.10.9.4.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := coth(x);
       writeln('coth(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich $\coth(x)$ die folgenden Einschließungen:

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e-30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
coth(x) enthalten in:
 1.000000000000E+030
 3.333333333333E-031
 3.333333333333E-044
 3.333333333333E-057
 3.333333333333E-070
 3.333333331111E-083
 1.111111111111E-096
 1.111111111111E-109
 1.111111111111E-122
 1.111111111111E-135
 1.111132275132E-148
 2.751322751322E-161
 7.513227513227E-174
 5.132275132275E-187
 1.322751322751E-200
 3.015873015873E-213
 1.587301587301E-227
 5.873015873015E-240
[  8.730158730158E-253,  8.730158730160E-253 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e-190 0 0 0 0 0 0       <enter>
coth(x) enthalten in:
 1.000000000000E+190
 3.333333333333E-191
 3.333333333333E-204
 3.333333333333E-217
 3.333333333333E-230
 3.333333333333E-243
[  3.333333333320E-256,  3.333333333340E-256 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
0.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0       <enter>
coth(x) enthalten in:
 2.163953413738E+000
 6.528487700040E-013
 1.021802311709E-026
 3.738602150792E-039
 2.725335741192E-052
 9.608763478333E-065
 9.486574409418E-078
 8.097501153092E-091
 4.047716140809E-104
 1.706658526910E-117
 3.838239019820E-130
 5.889548335892E-143
 3.228188086368E-156
 7.061468412602E-169
 7.504947417138E-182
 6.079091135747E-195
 9.934803284321E-208
 6.081398613958E-221
[  8.081533319760E-234,  8.081533319770E-234 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e50 0 0 0 0 0 0   <enter>
coth(x) enthalten in:
 1.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  3.000000000000E-267 ]
```

1.10.10 Die inversen hyperbolischen Funktionen

1.10.10.1 artanh(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen artanh–Funktion:

$$\operatorname{artanh}(x) := \{y \,|\, y = \operatorname{artanh}(\zeta),\ \zeta \in x \subseteq (-1, +1)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von artanh(x) erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.10.1.1 Der Algorithmus

Zunächst wird der Wertebereich artanh(x) mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\operatorname{artanh}(-x) \equiv -\operatorname{artanh}(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für artanh(x) eine Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x) < 1, \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Für den Wertebereich artanh(x) benutzen wir die Identitäten:

$$\operatorname{artanh}(x) = \frac{1}{2} \cdot \begin{cases} (\ln(1+x) - \ln(1-x)) & \inf(x) > 0.9999999999999 \\ \ln\left(\dfrac{1+x}{1-x}\right) & \inf(x) > 10^{-3} \\ \ln\left(1 + \dfrac{2x}{1-x}\right) & x \subset (0, 10^{-3}] \end{cases}$$

und mit den bereits implementierten Funktionen `ln` und `ln1p` gilt:

$$\begin{aligned} \frac{1}{2} \cdot \ln\left(\frac{1+x}{1-x}\right) &\subseteq 0.5 \diamond \mathtt{ln}(\,(1 \oplus x) \oslash (1 \ominus x)\,) \\ \frac{1}{2} \cdot \ln\left(1 + \frac{2x}{1-x}\right) &\subseteq 0.5 \diamond \mathtt{ln1p}(\,2 \diamond x \oslash (1 \ominus x)\,) \end{aligned}$$

Mit Hilfe der letzten Darstellung kann der Wertebereich artanh(x) auch für Argumentintervalle aus der unmittelbaren Umgebung des Ursprungs ohne Auslöschungseffekte eingeschlossen werden.

Als Alternative dazu bietet sich auch die Potenzreihenentwicklung der artanh–Funktion an:

$$\operatorname{artanh}(x) = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \ldots = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{2k+1}$$

Da die Reihe wegen der nur langsam abnehmenden Taylor–Koeffizienten jedoch relativ schlecht konvergiert, bevorzuge ich die Einschließung mit Hilfe der `ln1p(...)`–Funktion.

1.10.10.1.2 Der Programm-Quelltext

```
function artanh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
   c1   = 1e-3;
   c2   = 0.9999999999999;
var t        : Stagg[-1..Stagg_prec];
    d        : real;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   d := t[1] + t[-1];
   if grob then t := artanh( ival(t) )
           else  { t ist schmal, und es gilt: inf(t) > 0 }
    if d > c2 then t := 0.5*(ln(1+t)-ln(1-t)) { No Overflow }
              else if d > c1 then t := 0.5 * ln((1+t)/(1-t))
                             else t := 0.5 * ln1p(2*t/(1-t));
   if neg then t := -t;
   artanh := t;
end; { artanh }
```

1.10.10.1.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := artanh(x);
       writeln('artanh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich artanh(x) die folgenden Einschließungen:

```
Stagg_prec = ?
8      <enter>
x: Stagg = ?
8 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0 0 0    <enter>
artanh(x) enthalten in:
 1.000000000000E-020
 3.333333333333E-061
 3.333333333333E-074
 3.333333333333E-087
 3.533333333333E-100
 3.333333333333E-113
 3.333333333333E-126
 0.000000000000E+000
[  3.333333333332E-139,  3.347619047620E-139 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1e-267   <enter>
artanh(x) enthalten in:
 3.077416835049E+002
 8.507147111047E-011
 5.261091709998E-024
 7.847987991213E-037
 1.992750961478E-050
 3.917640202911E-063
 4.034693085585E-076
 4.295958217242E-089
 3.699910155499E-102
 5.354270144097E-115
 6.433767293465E-129
 6.991505048935E-142
 2.693347609687E-155
 5.735610242439E-168
 4.786140778402E-181
 7.609463380456E-194
 1.762835705230E-208
 4.697701619517E-221
[  8.387458425937E-234,  8.387458425954E-234 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
0.5 0 0 0 0 0 0    <enter>
artanh(x) enthalten in:
 5.493061443340E-001
 5.484569762261E-014
 8.461262852323E-027
 7.452789113747E-040
 2.586734716681E-053
 8.747146609304E-066
[  4.834368078760E-079,  4.834368078780E-079 ]
```

1.10.10.2 arcoth(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arcoth–Funktion:

$$\operatorname{arcoth}(x) := \{y \mid y = \operatorname{arcoth}(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)/[-1, +1]\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\operatorname{arcoth}(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.10.2.1 Der Algorithmus

Zunächst wird der Wertebereich $\operatorname{arcoth}(x)$ mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\operatorname{arcoth}(-x) \equiv -\operatorname{arcoth}(x)$ kann man sich daher beschränken auf $\inf(x) > 1$, so daß die eigentliche Aufgabe darin besteht, für $\operatorname{arcoth}(x)$ eine Einschließung zu berechnen unter der Voraussetzung:

$$1 < \inf(x) \leq \sup(x) < +\infty, \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Für den Wertebereich $\operatorname{arcoth}(x)$ benutzen wir die Identitäten:

$$\operatorname{arcoth}(x) = \frac{1}{2} \cdot \begin{cases} \ln\left(1 + \dfrac{2}{x-1}\right) & \inf(x) > 10^{+3} \\ (\ln(x+1) - \ln(x-1)) & \sup(x) < 1.000000000001 \\ \ln\left(\dfrac{x+1}{x-1}\right) & \text{sonst} \end{cases}$$

und mit den bereits implementierten Funktionen `ln` und `ln1p` gilt:

$$\begin{aligned} \frac{1}{2} \cdot \ln\left(\frac{x+1}{x-1}\right) &\subseteq \quad 0.5 \mathbin{\diamond} \mathtt{ln}(\,(x \mathbin{\diamond\!\!\!+} 1) \mathbin{\diamond\!\!\!/} (x \mathbin{\diamond\!\!\!-} 1)\,) \\ \frac{1}{2} \cdot \ln\left(1 + \frac{2}{x-1}\right) &\subseteq \quad 0.5 \mathbin{\diamond} \mathtt{ln1p}(\,2 \mathbin{\diamond\!\!\!/} (x \mathbin{\diamond\!\!\!-} 1)\,) \end{aligned}$$

$$\frac{1}{2} \cdot (\ln(x+1) - \ln(x-1)) \subseteq 0.5 \diamond (\mathbf{ln}(x \oplus 1) - \mathbf{ln}(x \ominus 1)$$

Diese aufwendige Darstellung mit den beiden ln–Funktionen wird benutzt, um für Intervalle x, die sehr dicht an der Polstelle 1 liegen, einen Overflow bei Auswertung des Quotienten $(x+1)/(x-1)$ zu vermeiden.

1.10.10.2.2 Der Programm-Quelltext

```
function arcoth(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
   c1   = 1e3;
   c2   = 1.000000000001;
var t        : Stagg[-1..Stagg_prec];
    d        : real;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;
   grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   d := t[1] + t[-1];
   if grob then t := arcoth( ival(t) )
           else  { t ist schmal, und es gilt: inf(t) > 0 }
     if d > c1 then t := 0.5 * ln1p( 2/(t-1) )
        else if d < c2 then t := 0.5 * ( ln(t+1) - ln(t-1) )
                       else t := 0.5 * ln( (t+1)/(t-1) );
   if neg then t := -t;
   arcoth := t;
end; { arcoth }
```

1.10.10.2.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := arcoth(x);
       writeln('arcoth(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich arcoth(x) die folgenden Einschließungen:

```
Stagg_prec = ?
8      <enter>
x: Stagg = ?
8 Stagg components and 1 interval are needed in this sequence
1e20 0 0 0 0 0 0 0 0   <enter>
arcoth(x) enthalten in:
 1.000000000000E-020
 3.333333333333E-061
 3.333333333333E-074
 3.333333333333E-087
 3.533333333333E-100
 3.333333333333E-113
 3.333333333333E-126
 0.000000000000E+000
[  3.333333333332E-139,  3.347619047620E-139 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1e-267    <enter>
arcoth(x) enthalten in:
 3.077416835049E+002
 8.507147111047E-011
 5.261091709998E-024
 7.847987991213E-037
 1.992750961478E-050
 3.917640202911E-063
 4.034693085585E-076
 4.295958217242E-089
 3.699910155499E-102
 5.354270144097E-115
 6.433767293465E-129
 6.991505048935E-142
 2.693347609687E-155
 5.735610242439E-168
 4.786140778402E-181
 7.609463380456E-194
 1.762835705230E-208
 4.697701619517E-221
[  8.387458425937E-234,  8.387458425954E-234 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
3 0 0 0 0 0 0    <enter>
arcoth(x) enthalten in:
 3.465735902799E-001
 7.265470861606E-014
 7.290882840377E-028
 5.006718012762E-041
 7.060340004746E-054
 6.968109848473E-067
[  5.780293166320E-080,  5.780293166375E-080 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   <enter>
arcoth(x) enthalten in:
 3.465735902799E-001
 7.265470861606E-014
 7.290882840377E-028
 5.006718012762E-041
 7.060340004746E-054
 6.968109848473E-067
 5.780293166349E-080
 8.209343771000E-093
 7.405102853428E-106
 6.684276011787E-119
 9.065278516335E-132
 3.758175379809E-145
 6.536378541418E-158
 5.717595153519E-171
 3.119458367355E-184
 6.167505768224E-197
 8.977619560237E-210
 5.863407874660E-223
[  3.257776236704E-236,  3.257776236709E-236 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e100 0 0 0 0 0 0  <enter>
arcoth(x) enthalten in:
 9.999999999999E-101
 9.999999999999E-114
 9.999999999999E-127
 9.999999999999E-140
 9.999999999999E-153
 9.999999999999E-166
[  9.999999999999E-179,  1.000000000001E-178 ]
```

1.10.10.3 arsinh(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arsinh–Funktion:

$$\mathrm{arsinh}(x) := \{y \,|\, y = \mathrm{arsinh}(\zeta),\ \zeta \in x \subseteq (-\infty, +\infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: x_j = `x[j]`, wobei die Bedingung $\mathrm{ub}(x)$ = `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter S = `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\mathrm{arsinh}(x)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter S = `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.10.3.1 Der Algorithmus

Zunächst wird der Wertebereich $\mathrm{arsinh}(x)$ mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn 0 in x enthalten ist oder wenn der relative Durchmesser von x größer ist als 10^{-14}. Wegen der Identität $\mathrm{arsinh}(-x) \equiv -\mathrm{arsinh}(x)$ kann man sich daher beschränken auf $\inf(x) > 0$, so daß die eigentliche Aufgabe darin besteht, für $\mathrm{arsinh}(x)$ eine Einschließung zu berechnen unter der Voraussetzung:

$$0 < \inf(x) \leq \sup(x), \quad \text{mit:} \quad \mathrm{diam}(x)/\inf(x) \leq 10^{-14}$$

Für den Wertebereich $\mathrm{arsinh}(x)$ gilt:

$$\mathrm{arsinh}(x) \subset \begin{cases} \mathrm{artanh}\left(\dfrac{x}{\sqrt{x^2+1}}\right) & \sup(x) < 10^{-5} \\ \ln(x) + \ln(2) + \left[0, \dfrac{1}{\inf(x)}\right] & \inf(x) \gg 1 \\ \ln(x + \sqrt{x^2+1}) & \text{sonst} \end{cases}$$

Die Auswertung mit der schon implementierten artanh–Funktion erfolgt mit der durch S = `Stagg_prec` vorgegebenen Präzision, während die Auswertung der ln–Funktionen mit der erhöhten Präzision $S+1$ vorgenommen wird.

Zu klären bleibt noch die Einschließung:

$$\operatorname{arsinh}(x) \subset \ln(x) + \ln(2) + \left[0, \frac{1}{\inf(x)}\right], \quad \inf(x) \gg 1 \tag{1.121}$$

Um (1.121) zu beweisen, betrachten wir die für $r > 0$ gültige Beziehung:

$$\begin{aligned}
0 &< \operatorname{arsinh}(r) - \ln(2r) \equiv \ln(r + \sqrt{r^2+1}) - \ln(2r) \\
&= \ln\left(\frac{r + \sqrt{r^2+1}}{2r}\right) = \ln\left(1 + \frac{\sqrt{1+1/r^2}-1}{2}\right) \\
&< \frac{\sqrt{1+1/r^2}-1}{2} < \frac{1}{r}
\end{aligned}$$

Für $r > 0$ gilt damit

$$\ln(2r) < \operatorname{arsinh}(r) < \ln(2r) + \frac{1}{r}, \qquad r > 0$$

woraus sich dann die Einschließung (1.121) direkt ergibt und umso enger ausfällt, je größer $\inf(x)$ gewählt wird. Im Programm wird (1.121) benutzt, wenn $\inf(x) > 10^{253}$, um Overflow zu vermeiden, der sonst bei Anwendung von $\operatorname{arsinh}(x) = \ln(x + \sqrt{x^2+1})$ auftreten würde. Beachten Sie bitte, daß dieser Overflow jedoch nicht entsteht bei Auswertung der Funktion

$$\texttt{sqrt1px2(x)} \supseteq \sqrt{1+x^2}$$

sondern bei Auswertung der Summe $\texttt{x} \mathbin{\diamond\!\!\!\!+} \texttt{sqrt1px2(x)} \supseteq x + \sqrt{1+x^2}$.

Eine mögliche Alternative zu

$$\operatorname{arsinh}(x) = \operatorname{artanh}\left(\frac{x}{\sqrt{x^2+1}}\right), \quad x \subset (0, 10^{-5}]$$

ist noch die für $\sup(x) < 1$ gültige Potenzreihe:

$$\begin{aligned}
\operatorname{arsinh}(x) &= x - \frac{x^3}{6} + \frac{3x^5}{40} - \frac{15x^7}{336} \pm \ldots \\
&= x + \sum_{k=1}^{\infty} \frac{1 \cdot 3 \cdot 5 \cdot \ldots \cdot (2k-1)(-1)^k \cdot x^{2k+1}}{2 \cdot 4 \cdot 6 \cdot \ldots \cdot (2k)(2k+1)}
\end{aligned}$$

deren Auswertung jedoch nicht wesentlich schneller ist als die Anwendung der schon implementierten artanh–Funktion.

1.10.10.3.2 Der Programm-Quelltext

```
function arsinh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c = 1e-14;
   c1   = 1e-5;
var t        : Stagg[-1..Stagg_prec];
   t1        : Stagg;
    d        : real;
   xi        : interval;
   S         : integer;
   neg,grob : boolean;
begin
   t := x;  Normal(t);  neg := false;  grob := 0 in t;
   if not grob then
   begin
      d := t[0] -> t[-1];  { d: Intervalldurchmesser }
      neg := SIGN(t[1]+t[-1]) < 0;
      if neg then t := abs(t);
      grob := d /< (t[1]+t[0]) > c;
   end;
   if grob then t := arsinh( ival(t) )
           else  { t ist schmal, und es gilt: inf(t) > 0 }
     if t < c1 then t := artanh( t/sqrt1px2(t) )
     else
     begin
        S := get_Stagg_prec;
        allocate(t1,-1..S+1);  set_Stagg_prec(S+1);
        t1 := change_length(t);
        if expo(t[1]+t[0]) > 254 then
        begin
           xi := 1/ival(t);
           t1 := ln(t1) + ln(Staggintval(2)) +
           Staggintval(0,xi.sup);
        end    else t1 := ln( t1 + sqrt1px2(t1) );
        set_Stagg_prec(S);
        t := change_length(t1);  free(t1);
     end;
   if neg then t := -t;
   arsinh := t;
end; { arsinh }
```

1.10.10.3.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := arsinh(x);
       writeln('arsinh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich arcosh(x) die folgenden Einschließungen:

```
Stagg_prec = ?
8      <enter>
x: Stagg = ?
8 Stagg components and 1 interval are needed in this sequence
9.999999999999e255 0 0 0 0 0 0 0 0    <enter>
arsinh(x) enthalten in:
 5.901549309870E+002
 3.554041802304E-011
 4.515655413713E-024
 9.571478899928E-037
 3.760319289323E-051
 8.566483660385E-064
 9.615602114782E-077
 9.919297348800E-090
[  3.866013156462E-104,  3.866013156463E-104 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
arsinh(x) enthalten in:
 1.818446459232E+000
 6.682348369896E-014
 3.560708993786E-027
 2.539427681216E-040
 1.745174416723E-053
 3.054107866175E-066
 7.510260840443E-079
 6.079269363084E-092
 9.194688453264E-106
 9.219086776276E-119
 7.830101950665E-132
 9.896630731977E-145
 8.699563022006E-158
 7.845359607428E-171
 9.182698039282E-184
 3.261333202564E-197
 3.513294096753E-210
 9.469024723486E-223
[  3.908478847872E-236,  3.908478847873E-236 ]

Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e-20 0 0 0 0 0 0      <enter>
arsinh(x) enthalten in:
 9.999999999999E-021
 9.999999999999E-034
 9.999999999999E-047
 9.833333333333E-060
 3.333333333333E-073
 3.333333333333E-086
[  3.340833333333E-099,  3.340833333334E-099 ]
```

1.10.10.4 arcosh(x)

Zu gegebenem staggered Argument $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ ist einzuschließen der Wertebereich der reellen arcosh–Funktion:

$$\operatorname{arcosh}(x) := \{y \,|\, y = \operatorname{arcosh}(\zeta),\ \zeta \in x \subseteq [+1, \infty)\}$$

Im Programm wird x realisiert durch eine Variable `x` vom Typ `Stagg`, und es gilt: $x_j = \mathtt{x}[\mathtt{j}]$, wobei die Bedingung $\mathrm{ub}(x) =$ `Stagg_prec` erfüllt sein muß. Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von arcosh(x) erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. Der Parameter $S =$ `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

1.10.10.4.1 Der Algorithmus

Zunächst wird der Wertebereich arcosh(x) mit Hilfe der schon implementierten Intervallfunktion im einfachen `real`–Format eingeschlossen, wenn der relative Durchmesser von x größer ist als 10^{-14}. Danach erfolgt die Einschließung des Wertebereichs arcosh(x) durch:

$$\operatorname{arcosh}(x) \subset \begin{cases} \ln(x) + \ln(2) + \left[\dfrac{-1}{\inf(x)}, 0\right] & \inf(x) \gg 1 \\ \ln(x + \sqrt{x^2 - 1}) & \inf(x) > 1.01 \\ \ln\left[1 + \sqrt{x-1} \cdot (\sqrt{x-1} + \sqrt{x+1})\right] & x \subset [1, 1.01] \end{cases}$$

dabei werden die Wertebereiche $\ln(x)$ bzw. $\sqrt{x^2 - 1}$ auf der Maschine mit den bereits implementierten staggered Funktion `ln(x)` bzw. `sqrtx2m1(x)` eingeschlossen. Für die Intervalle $x \subset [1, 1.01]$ gilt mit Hilfe der ebenfalls schon implementierten `ln1p(...)`-Funktion:

$$\ln\left[1 + \sqrt{x-1} \cdot (\sqrt{x-1} + \sqrt{x+1})\right] \subseteq$$
$$\mathtt{ln1p(sqrt(x} \diamondminus \mathtt{1)} \diamonddot \mathtt{(sqrt(x} \diamondminus \mathtt{1)} \diamondplus \mathtt{sqrt(x} \diamondplus \mathtt{1)))}$$

Diese letzte Einschließung ist wegen der zwei verschiedenen Wurzelberechnungen zwar etwas aufwendig, aber Berechnungen von arcosh$(1 + \varepsilon)$ können damit für $\varepsilon \ll 1$ sehr genau ausgeführt werden!

Zu klären bleibt noch die Einschließung:

$$\operatorname{arcosh}(x) \subset \ln(x) + \ln(2) + \left[\frac{-1}{\inf(x)}, 0\right], \quad \inf(x) \gg 1 \tag{1.122}$$

Um (1.122) zu beweisen, betrachten wir die für $r > 1$ gültige Beziehung:

$$\begin{aligned} 0 &< \ln(2r) - \operatorname{arcosh}(r) \equiv \ln(2r) - \ln(r + \sqrt{r^2-1}) \\ &= \ln\left(\frac{2r}{r+\sqrt{r^2-1}}\right) = \ln\left(1 + \frac{r-\sqrt{r^2-1}}{r+\sqrt{r^2-1}}\right) \\ &< \frac{r-\sqrt{r^2-1}}{r+\sqrt{r^2-1}} < \frac{1/r}{r+\sqrt{r^2-1}} < \frac{1/r}{2r-1/r} = \frac{1}{2r^2-1} < \frac{1}{r} \end{aligned}$$

Für $r > 1$ gilt damit

$$\ln(2r) - \frac{1}{r} < \operatorname{arcosh}(r) < \ln(2r), \qquad r > 1$$

woraus sich dann die Einschließung (1.122) direkt ergibt und umso enger ausfällt, je größer $\inf(x)$ gewählt wird. Im Programm wird (1.122) benutzt, wenn $\inf(x) > 10^{253}$, um Overflow zu vermeiden, der sonst bei Anwendung von $\operatorname{arcosh}(x) = \ln(x + \sqrt{x^2-1})$ auftreten würde. Beachten Sie bitte, daß dieser Overflow jedoch nicht entsteht bei Auswertung der Funktion

$$\texttt{sqrtx2m1(x)} \supseteq \sqrt{x^2-1}$$

sondern bei Auswertung der Summe $\texttt{x} \oplus \texttt{sqrtx2m1(x)} \supseteq x + \sqrt{x^2-1}$.

Eine mögliche Alternative zu

$$\operatorname{arcosh}(x) = \ln\left[1 + \sqrt{x-1}\cdot(\sqrt{x-1} + \sqrt{x+1})\right], \quad x \subset [1, 1.01]$$

ist noch die Darstellung:

$$\begin{aligned} &c_0 = 1, \quad c_{k+1} = -c_k \cdot \frac{k+1}{2k+3}, \quad k = 0,1,2,\ldots, \quad P(h) := \sum_{k=0}^{\infty} c_k \cdot h^k, \\ &\leadsto \quad \operatorname{arcosh}(x) = \sqrt{x^2-1}\cdot P(h), \quad 0 \le h \equiv x - 1 < 1 \end{aligned}$$

Da die Reihe $P(h)$ jedoch nur mäßig konvergiert, habe ich die Darstellung mit Hilfe der Logarithmus–Funktion bevorzugt.

1.10.10.4.2 Der Programm-Quelltext

```
function arcosh(var x : Stagg) : Stagg[-1..Stagg_prec];
const c =  1e-14;
var t     : Stagg[-1..Stagg_prec];
   t1,t2 : Stagg;
   d     : real;
   xi    : interval;
   S     : integer;
begin
   t := x;   Normal(t);
   d := t[0] -> t[-1];  { d: Intervalldurchmesser }
   if d /< (t[1]+t[0]) > c then arcosh := arcosh(ival(t))
   else
   begin
      S := get_Stagg_prec;  allocate(t1,-1..S+1);
      set_Stagg_prec(S+1);
      t1 := change_length(t);
      if t > 1.01 then
      begin
         if expo(t1[1]+t1[-1]) > 254 then
         begin  { Approximation:   arcosh(x) = ln(2)+ln(x) }
            xi := -1/ival(t1);
            t1 := ln(t1) + ln(Staggintval(2))
            + Staggintval(xi.inf,0)
         end     else t1 := ln( t1+sqrtx2m1(t1) );
      end    else
      begin
         allocate(t2,-1..S+1);
         t2 := sqrt(t1-1);
         t1 := ln1p(t2 * (t2 + sqrt(t1+1)));  free(t2);
      end;
      set_Stagg_prec(S);
      t := change_length(t1);  free(t1);
      if SIGN(t[1]+t[-1]) < 0 then
         t := hull(Staggintval(0),sup(t));
      arcosh := t;
   end;
end; { arcosh }
```

1.10.10.4.3 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       y := arcosh(x);
       writeln('arcosh(x) enthalten in: ');
       write(y);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich arcosh(x) die folgenden Einschließungen:

```
Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
9.999999999999e255 0 0 0 0 0 0      <enter>
arcosh(x) enthalten in:
 5.901549309870E+002
 3.554041802304E-011
 4.515655413713E-024
 9.571478899928E-037
 3.760319289323E-051
 8.566483660385E-064
 9.615602114782E-077
[  9.919297348800E-090,  9.919297348801E-090 ]
```

```
Stagg_prec = ?
17     <enter>
x: Stagg = ?
17 Stagg components and 1 interval are needed in this sequence
3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    <enter>
arcosh(x) enthalten in:
 1.762747174039E+000
 8.605046521864E-014
 9.959584618056E-027
 3.206565232708E-040
 2.150659121730E-053
 6.754368444052E-066
 1.756674137838E-079
 2.051208571347E-092
 9.632384212984E-105
 3.775241450239E-118
 5.318387505451E-131
 9.255315808184E-145
 4.315736072579E-158
 4.392480614714E-171
 8.192510979557E-184
 4.312652473561E-197
 3.013526065790E-210
[  8.083271163801E-223,  8.083271163802E-223 ]

Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
1 1e-100 1e-120 1e-140 1e-160 1e-180 1e-200 1e-267    <enter>
arcosh(x) enthalten in:
 1.414213562373E-050
 9.504880875979E-064
 2.021563553813E-077
 7.333520065855E-090
 5.463385860263E-103
 6.513324098139E-116
 5.546303843574E-129
[  1.727756053203E-142,  1.727756053204E-142 ]
```

```
Stagg_prec = ?
17     <enter>
x: Stagg = ?
17 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1e-267      <enter>
arcosh(x) enthalten in:
 4.472135954999E-134
 5.793928183473E-147
 3.746255247088E-160
 1.236719223051E-173
 4.485417944908E-186
 2.104185127560E-199
 9.798828828816E-212
 7.575645499390E-225
 1.635230154756E-238
 7.008506535448E-251
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  8.936000000000E-264,  8.946000000000E-264 ]

Stagg_prec = ?
7      <enter>
x: Stagg = ?
7 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 [0,1e-267]  <enter>
arcosh(x) enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  4.472135955005E-134 ]
```

Kapitel 2

Standardfunktionen mit zwei Variablen

In diesem Kapitel werden die Algorithmen vorgestellt, mit denen die im Buch

Verifizierende Numerik mit
PASCAL-XSC *BCD-VERSION*
NUMERISCHE ANWENDUNGEN
HILFS-MODULE BIBLIOTHEK1

angegebenen reellen Funktionen mit zwei Variablen in *interval staggered arithmetic* realisiert wurden. Wir beschränken uns dabei auf den Basistyp `real`, da die Algorithmen zum Basistyp `Longreal` bis auf die entsprechenden internen Konstanten identisch sind. Das Hauptanwendungsgebiet dieser reellen Funktionen ist die Realisierung von Real- bzw. Imaginär–Teil einiger komplexer Standardfunktionen.

Die Funktionen zum Basistyp `real` bzw. `Longreal` werden von den Hilfsmodulen `s_fu1` bzw. `ls_fu1` exportiert. Weitere programmiertechnische Einzelheiten findet man in dem oben genannten Buch und in den vielen Programm-Beispielen dieses Kapitels.

Zur Deklaration einer *staggered* Variablen und zur Steuerung der Präzision der *staggered* Arithmetik vergleichen Sie bitte die Anmerkungen von Seite 7.

2.1 $\sqrt{x^2+y^2}$

Zu gegebenen staggered Intervall–Argumenten x, y ist einzuschließen der Wertebereich:

$$\sqrt{x^2+y^2} := \left\{ t \,|\, t = \sqrt{\zeta^2+\xi^2},\ \zeta \in x,\ \xi \in y;\ \ x,y \subset (-\infty,+\infty) \right\}$$

Im Programm wird dabei das Intervall $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ realisiert durch eine Variable `x : Stagg`. Dabei gilt: $x_j =$ `x[j]`, wobei die Bedingung $\mathrm{ub}(x) = \mathrm{ub}(y) =$ `Stagg_prec` erfüllt sein muß. Das Intervall `y` wird ganz analog definiert. Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\sqrt{x^2+y^2}$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter $S =$ `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

2.1.0.4.4 Der Algorithmus

Zunächst wird der Fall $x = 0$ oder $y = 0$ abgefragt. Im Falle $x = 0$ gilt beispielsweise

$$\sqrt{x^2+y^2} = |y|, \quad \text{falls } \ x = 0;$$

Mit dem Aufruf der Funktion

```
function grob_calc(var x,y : Stagg) : boolean;
```

wird dann entschieden, ob $\sqrt{x^2+y^2}$ mit Hilfe der bereits implementierten Funktion `sqrtx2y2(ival(x),ival(y))` im einfachen `real`–Format eingeschlossen wird, oder ob der Wertebereich $\sqrt{x^2+y^2}$ durch Anwendung der staggered Arithmetik einzuschließen ist. Weitere Einzelheiten findet man im Quelltext auf Seite 211.

Ist dann x das schmale Intervall, dessen Elemente mindestens die gleiche Größenordnung besitzen wie die Elemente aus y, so wird für $\sqrt{x^2+y^2}$ die folgende Identität benutzt:

$$\sqrt{x^2+y^2} = |x| \cdot \sqrt{1 + \left(\frac{y}{x}\right)^2}$$

2.1.0.4.5 Der Programm-Quelltext

```
function grob_S(var x : Stagg; nullinx : boolean) : boolean;
{ grob_S entscheidet, ob mit x grob gerechnet wird.  }
{ Vorauss.: inf(x) >= 0;  nullinx = TRUE <==> 0 in x }
var bl : boolean;
begin
   bl := nullinx;
   if not bl then
      if x[0] <> x[-1] then
         bl := expo(x[0]->x[-1]) - expo(x[1]+>x[-1]) > -14;
   grob_S := bl;
end; { grob_S }

function grob_calc(var x,y : Stagg) : boolean;
{ grobcalc entscheidet, ob mit x,y grob gerechnet wird. }
{ Vorauss.:  inf(x) >= 0,  inf(y) >= 0;                 }
{ Vorsicht:  x,y  koennen vertauscht werden!            }
var nullinx,nulliny,grob,bl : boolean;
   t                        : Stagg[-1..Stagg_prec];
   exx,exy,p                : integer;
begin
   nullinx := 0 in x;  nulliny := 0 in y;
   grob := nullinx and nulliny;
   if not grob then
   begin
      exx := expo(x[1]+x[0]);   exy := expo(y[1]+y[0]);
      if exy > exx then
      begin
         t := x;  x := y;  y := t;
         p := exx;  exx := exy;  exy := p;
         bl := nullinx;  nullinx := nulliny;  nulliny := bl;
      end;
      bl := exx - exy < 14;
      grob := grob_S(x,nullinx) or (bl and grob_S(y,nulliny));
   end;
   grob_calc := grob;
end; { grob_calc }
```

```
function sqrtx2y2(var x,y : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet Einschliessung fuer:   sqrt(x^2+y^2) }
var u,v,t  : Stagg[-1..Stagg_prec];
   exu,exv : integer;
   grob    : boolean;
begin
   u := abs(x);   v := abs(y);
   if u=0 then t := v else
      if v=0 then t := u else
   begin
      grob := grob_calc(u,v);
      if grob then t := sqrtx2y2(ival(u),ival(v))
      else
      begin { u ist schmal und es gilt:  inf(u) > 0 }
         exu := expo(u[1] + u[-1]);
         exv := expo(v[1] + v[0]);
         if exv > exu then
         begin
            t := u;  u := v;  v := t;
            exu := exv;
         end;
         { u ist das groessere der Intervalle u,v }
         v := v/u;
         t := u*sqrt(1+v*v);  { sqrt(x^2+y^2) in t enthalten }
      end;
   end;
   sqrtx2y2 := t;
end; { sqrtx2y2 }
```

Bitte beachten Sie, daß im obigen Programm der Wertebereich rechts in

$$\sqrt{x^2+y^2} = |x| \cdot \sqrt{1+\left(\frac{y}{x}\right)^2}$$

durch die obigen Anweisungen

```
v := v/u;
t := u*sqrt(1+v*v);  { sqrt(x^2+y^2) in t enthalten }
```

intervallmäßig ausgewertet wird, womit dann die garantierte Einschließung von $\sqrt{x^2+y^2} \subseteq \texttt{t}$ gewährleistet ist.

2.1.0.4.6 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y,z : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       writeln('y: Stagg = ?');  read(y);
       z := sqrtx2y2(x,y);
       writeln('sqrt(x^2+y^2) enthalten in: ');
       write(z);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');  read(p);
   set_Stagg_prec(p);
   main;
end.
```

liefert für den Wertebereich $\sqrt{x^2+y^2}$ die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e255 0 0 0 0 0 0  <enter>
y: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e255 0 0 0 0 0 0  <enter>
sqrt(x^2+y^2) enthalten in:
 1.414213562373E+255
 9.504880168872E+241
 4.209698078569E+228
 6.718753769480E+215
 7.317667973799E+202
 7.324784621070E+188
[  3.885038753200E+175,  3.885038753500E+175 ]
```

```
Stagg_prec = ?
2      <enter>
x: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-2,3]   <enter>
y: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
1e250 0 0    <enter>
sqrt(x^2+y^2) enthalten in:
 1.000000000000E+250
 0.000000000000E+000
[ -0.000000000000E+000,  1.000000000000E-017 ]

Stagg_prec = ?
3      <enter>
x: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
0 0 0 0    <enter>
y: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
22 0 0 [0,5]  <enter>
sqrt(x^2+y^2) enthalten in:
 0.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[  2.200000000000E+001,  2.700000000000E+001 ]

Stagg_prec = ?
3      <enter>
x: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
4 0 0 0    <enter>
y: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
3 0 0 0    <enter>
sqrt(x^2+y^2) enthalten in:
 5.000000000000E+000
 0.000000000000E+000
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
15 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0   <enter>
y: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1e-30 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 [0,1e-267]     <enter>
sqrt(x^2+y^2) enthalten in:
 1.500000000000E+001
 3.333333333333E-062
 3.333333333333E-075
 3.333333333333E-088
 3.333333333333E-101
 3.333333333296E-114
 2.962962962962E-127
 9.629629629629E-140
 6.296296296296E-153
 2.962962962962E-166
 9.629629637860E-179
 8.230452674897E-193
 1.193415637860E-206
 8.230452674897E-220
 1.193415637860E-233
 8.228166438042E-247
 0.000000000000E+000
 0.000000000000E+000
[  9.812528000000E-260,  9.812531000000E-260 ]

Stagg_prec = ?
1      <enter>
x: Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 0     <enter>
y: Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 0     <enter>
sqrt(x^2+y^2) enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

2.2 $\ln(\sqrt{x^2+y^2}\,)$

Zu gegebenen staggered Intervall–Argumenten x, y ist einzuschließen der Wertebereich:

$$\ln(\sqrt{x^2+y^2}\,) := \left\{ t \mid t = \ln(\sqrt{\zeta^2+\xi^2}\,),\ \zeta \in x,\ \xi \in y;\ \ x, y \subset (-\infty, +\infty) \right\}$$

Im Programm wird dabei das Intervall $x = x_1 + x_2 + \ldots + x_{\mathrm{ub}(x)} + [x_{-1}, x_0]$ realisiert durch eine Variable `x : Stagg`. Dabei gilt: $x_j = \mathtt{x[j]}$, wobei die Bedingung $\mathrm{ub}(x) = \mathrm{ub}(y) = \mathtt{Stagg\_prec}$ erfüllt sein muß. Das Intervall `y` wird ganz analog definiert. Der `integer`-Parameter $S = \mathtt{Stagg\_prec} \geq 1$ bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von $\ln(\sqrt{x^2+y^2}\,)$ erfolgt mit einer Präzision von mindestens $13 \cdot (S+1)$ Dezimalstellen. Der Parameter $S = \mathtt{Stagg\_prec}$ wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

2.2.0.4.7 Der Algorithmus

Nach Umrechnung der Intervalle x, y auf ihre Beträge wird zunächst der Fall $x = 0$ oder $y = 0$ abgefragt. Im Falle $x = 0$ gilt dann beispielsweise

$$\ln(\sqrt{x^2+y^2}\,) = \ln(|y|), \quad \text{falls } \ x = 0;$$

Im nächsten Schritt werden dann die Intervalle x, y so sortiert, daß x die größten Elemente enthält. Für den Wertebereich gelten dann die folgenden Identitäten:

$$\ln(\sqrt{x^2+y^2}) = \begin{cases} \ln(x) + \frac{1}{2} \cdot \ln\left[1 + \left(\frac{y}{x}\right)^2\right] & \inf(x) > 10^{+254} \\ \ln(\sqrt{x^2+y^2}\,) & \inf(x) > 10^{+126} \\ \ln(x) + \frac{1}{2} \cdot \ln\left[1 + \left(\frac{y}{x}\right)^2\right] & \sup(x) < 10^{-15} \\ \frac{1}{2} \cdot \ln(1 + (x^2+y^2-1)) & x^2+y^2 \subset [0.9, 1.1] \\ \frac{1}{2} \cdot \ln(x^2+y^2) & \text{sonst} \end{cases}$$

Die beiden ersten Identitäten werden benutzt, um Overflow zu vermeiden, wobei der Wertebereich $\sqrt{x^2+y^2}$ durch die bereits realisierte Funktion `sqrtx2y2(x,y)` eingeschlossen wird. Die obige vierte Identität kommt zur Anwendung, wenn x^2+y^2 in der unmittelbaren Nähe der 1 liegt oder diese Nullstelle der ln–Funktion enthält. Die `ln1p`–Funktion liefert dann mit dem Argument $(\mathtt{x} \diamond \mathtt{x} \,\diamond\!\!\!\!+\, \mathtt{y} \diamond \mathtt{y} \,\diamond\!\!\!\!-\, 1)$ gute Einschließungen von $\ln(\sqrt{x^2+y^2}\,)$, und es gilt wegen der Intervallauswertung dieser vierten Identität:

$$\ln(\sqrt{x^2+y^2}) \subseteq 0.5 \diamond \mathtt{ln1p}(\,\mathtt{x} \diamond \mathtt{x} \oplus \mathtt{y} \diamond \mathtt{y} \ominus 1\,).$$

Die dritte Identität kommt zur Anwendung, wenn die von Null verschiedenen Intervalle x, y in der unmittelbaren Umgebung des Ursprungs liegen. Die intervallmäßige Auswertung der beiden ln–Funktionen ist zwar aufwendig, liefert jedoch für den Wertebereich $\ln(\sqrt{x^2+y^2})$ sehr gute Einschließungen! Falls notwendig entscheiden wir uns also auf Kosten der Laufzeit für bessere Einschließungen. Beachten Sie bitte, daß bei der intervallmäßigen Auswertung von $\ln(x^2+y^2)$ mit von Null verschiedenen aber zu dicht am Ursprung liegenden Intervallen x, y die Einschließung $x \diamond x \oplus y \diamond y \supseteq x^2+y^2$ wegen der bekannten Intervallüberschätzungen selbst die Null enthalten kann, was dann bei Auswertung der ln–Funktion zu einer Fehlermeldung führt, die jedoch mit Hilfe der aufwendigeren dritten Identität vermieden wird!

2.2.0.4.8 Der Programm-Quelltext

```
function ln_sqrtx2y2(var x,y : Stagg) : Stagg[-1..Stagg_prec];
{ Berechnet Einschliessung von  ln[ sqrt(x^2+y^2) ] }
  function logg(var x : Stagg) : Stagg[-1..Stagg_prec];
  begin
     if x[1] = 1 then logg := ln1p(x-1) else logg := ln(x);
  end; { logg }
var u,v,t             : Stagg[-1..Stagg_prec];
   exu,exv,p          : integer;
begin
   u := abs(x);  v := abs(y);  Normal(u);  Normal(v);
   if u=0 then t := logg(v) else
      if v=0 then t := logg(u) else
      begin  { u,v <> 0, aber u oder v koennen 0 enthalten! }
         { Jetzt wird so sortiert, dass u die groessten     }
         { Elemente besitzt:                                }
         exu := expo(u[1]+u[0]);   exv := expo(v[1]+v[0]);
         if exv > exu then
         begin
            p := exu;  exu := exv;  exv := p;
            t := u;    u := v;      v := t;
         end;  { Sortierung beendet. }
         if exu > 126 { Vermeidet Overflow }  then
            if exu > 254 then
```

```
            begin  { Vermeidet ebenfalls Overflow }
               v := v/u;   t := ln(u) + 0.5*ln1p(v*v);
            end else t := ln( sqrtx2y2(u,v) ) else
            { u*u + v*v jetzt ohne Overflow berechenbar: }
            if exu < -15 then
            begin  { Falls 0 in u, dann Fehlermeldung }
               v := v/u;  t := ln(u) + 0.5*ln1p(v*v);
            end         else
            begin
               t := u*u + v*v;
               if (t[1]+t[-1]>0.9) and (t[1]+t[0]<1.1) then
                  t := 0.5 * ln1p(t-1) else t := 0.5 * ln(t);
            end;
      end;
   ln_sqrtx2y2 := t;
end; { ln_sqrtx2y2 }
```

2.2.0.4.9 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y,z : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       writeln('y: Stagg = ?');  read(y);
       z := ln_sqrtx2y2(x,y);
       writeln('ln( sqrt(x^2+y^2) ) enthalten in: ');
       write(z);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');   read(p);
   set_Stagg_prec(p);  main;
end.
```

liefert für den Wertebereich $\sqrt{x^2+y^2}$ die folgenden Einschließungen:

```
Stagg_prec = ?
6      <enter>
x: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0 0        <enter>
y: Stagg = ?
6 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0   <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
-6.147902198294E+002
-1.019763280371E-011
-8.400725243429E-024
-4.940974638823E-037
-8.460089854955E-050
-8.341886783853E-063
[ -1.122230112538E-076, -1.122230112537E-076 ]

Stagg_prec = ?
10     <enter>
x: Stagg = ?
10 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0 0 0 0 0   <enter>
y: Stagg = ?
10 Stagg components and 1 interval are needed in this sequence
1e-267 0 0 0 0 0 0 0 0 0 0   <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
-6.144436462391E+002
-3.022497809510E-011
-2.339996155145E-024
-4.563473967022E-037
-5.697383820955E-050
-3.595189972868E-063
-2.648652083220E-076
-9.542453631353E-089
-4.397598780143E-102
-4.225350053905E-116
[ -9.230338751384E-129, -9.230338751373E-129 ]
```

```
Stagg_prec = ?
18     <enter>
x: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
0.6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1e-30   <enter>
y: Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
0.8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1e-30  <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
-1.999999999999E-031
-9.999999999999E-044
-9.999039999999E-057
-9.999999999999E-070
-9.999999961066E-083
-6.666666666666E-096
-6.666666666675E-109
-9.866666666666E-123
-6.666666666666E-136
-6.740369066666E-149
-6.666666666666E-162
-6.666657880746E-175
-6.666666666666E-188
-6.666666665312E-201
-5.436952380952E-214
-3.809523809523E-227
-8.939183055238E-240
-9.523809523809E-254
[ -7.000000000000E-267, -4.000000000000E-267 ]
Stagg_prec = ?
2      <enter>
x: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0.6 0 1e-200   <enter>
y: Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0.8 0 0        <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
 5.999999999999E-201
 9.999999999999E-214
[  9.999999999999E-227,  1.000000000001E-226 ]
```

```
Stagg_prec = ?
3      <enter>
x: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
1 0 0 1e-30   <enter>
y: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
0 0 0 0       <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
 9.999999999999E-031
 9.999999999999E-044
 9.999500000000E-057
[  3.333333333333E-091,  3.333333333334E-091 ]

Stagg_prec = ?
16     <enter>
x: Stagg = ?
16 Stagg components and 1 interval are needed in this sequence
9.999e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
y: Stagg = ?
16 Stagg components and 1 interval are needed in this sequence
9.999e255 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
 5.898082573917E+002
 5.533440497909E-011
 4.959912038465E-024
 5.228957584114E-037
 3.841607819278E-050
 6.682568543630E-063
 4.708956011936E-076
 7.044903625278E-089
 5.999276444892E-102
 7.105479245533E-115
 8.285650982046E-128
 2.769249210312E-141
 8.523433090125E-154
 7.164370828134E-167
 7.009294795392E-180
 8.276180591161E-193
[  7.357807973686E-206,  7.357807973687E-206 ]
```

```
Stagg_prec = ?
16     <enter>
x: Stagg = ?
16 Stagg components and 1 interval are needed in this sequence
25 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
y: Stagg = ?
16 Stagg components and 1 interval are needed in this sequence
42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0     <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
 3.889315073662E+000
 9.047668192618E-013
 3.314419530133E-026
 8.699166061115E-039
 9.010346538395E-052
 8.236917915224E-066
 9.927745048510E-079
 9.450635752657E-092
 1.094911349111E-105
 1.986985223514E-118
 5.278685870116E-131
 1.420053109603E-144
 3.960543605154E-157
 4.866912961840E-170
 2.843965792725E-183
 9.433869063345E-196
[  2.005514452394E-209,  2.005514452397E-209 ]

Stagg_prec = ?
3      <enter>
x: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
0.22 0 0 0          <enter>
y: Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
0.97549987 0 0 0   <enter>
ln( sqrt(x^2+y^2) ) enthalten in:
-1.814991553294E-009
-1.943345433156E-022
-7.741681393098E-035
[ -8.740184819666E-048, -8.740184819664E-048 ]
```

2.3 arg(x, y)

Zu gegebenen staggered Intervall–Argumenten x, y ist einzuschließen der Wertebereich arg(x, y) aller Winkel α, welche durch die komplexen Zahlen $\zeta + i \cdot \eta$ definiert sind, mit $\zeta \in x$ und $\eta \in y$. Im Falle $0 \in x$ und $0 \in y$ erfolgt eine Fehlermeldung, da der komplexen Zahl $\zeta + i \cdot \eta = 0$ kein eindeutiger Winkel zugeordnet werden kann.

$$\arg(x, y) := \{t \,|\, t = \arg(\zeta, \eta),\ \zeta \in x,\ \eta \in y;\ \ x, y \subset (-\infty, +\infty)\}$$

Im Programm wird dabei das Intervall $x = x_1 + x_2 + \ldots + x_{\text{ub}(x)} + [x_{-1}, x_0]$ realisiert durch eine Variable `x : Stagg`. Dabei gilt: $x_j =$ `x[j]`, wobei die Bedingung $\text{ub}(x) = \text{ub}(y) =$ `Stagg_prec` erfüllt sein muß. Das Intervall `y` wird ganz analog definiert. Der `integer`-Parameter $S =$ `Stagg_prec` ≥ 1 bestimmt die Präzision der Rechnung, d.h. die interne Auswertung des Algorithmus zur Einschließung von arg(x, y) erfolgt mit einer Präzision von mindestens $13 \cdot (S + 1)$ Dezimalstellen. Der Parameter $S =$ `Stagg_prec` wird vom Modul `type_def` exportiert und kann vom Anwender mit Hilfe der Prozedur `set_Stagg_prec(prec : integer)` festgelegt werden.

2.3.0.4.10 Der Algorithmus

Das komplexe Intervall $z = x + i \cdot y$ kann interpretiert werden als Rechteck in der x, y-Ebene. Liegt nun z ganz im dritten Quadranten und gilt $\sup(y) = 0$, so sind alle Winkel $\alpha < 0$, und es gilt $-\pi \leq \alpha$. Liegt jedoch z in der linken Halbebene und die negative reelle Achse ganz im Innern von z, so sind alle Winkel $\alpha > \pi/2$, und die Oberschranke aller α ist größer als π aber kleiner als $3\pi/2$.

In der numerischen Praxis werden die relativen Durchmesser der beiden Intervalle x, y sehr klein sein. Wenn in seltenen Ausnahnefällen diese Voraussetzung jedoch nicht erfüllt ist, wird man verlangen, daß der Algorithmus auch dann garantierte und möglichst enge Einschließungen aller Winkel α liefert. Dies erreicht man dadurch, daß man ganz am Anfang die Intervalle x, y mit Hilfe der Funktionen `ival(x),ival(y)` durch Intervalle im groben `real`–Format einschließt und mit diesen Intervallen vom Typ `interval` die bereits implementierte Funktion `xi := arg(ival(x),ival(y))` aufruft. Ist dann `diam(xi)/sup(abs(xi))` $> 10^{-11}$, so erhält man mit `t := xi` die gesuchte grobe Einschließung, andernfalls wird die Einschließung der Winkel mit Hilfe der staggered Arithmetik in der duch $S =$ `Stagg_prec` ≥ 1 vorgegebenen Präzision durchgeführt.

Liegt das Intervall z dabei im ersten Quadranten, mit $\inf(x) > 0$, so gilt für die Winkel α aller Punkte aus z:

$$\alpha \in \arctan(y/x) \subseteq \mathtt{arctan}(y \diamondsuit x),$$

wobei `arctan` die bereits implementierte arctan–Funktion für staggered Argumente ist. Um Overflow bei der Berechnung von $(y \diamondsuit x)$ zu vermeiden, wird die Einschließung von α wie folgt berechnet

$$\alpha \in \frac{\pi}{2} - \arctan(x/y) \subseteq \langle \pi/2 \rangle \diamondsuit \mathtt{arctan}(x \diamondsuit y),$$

wenn z oberhalb der ersten Winkelhalbierenden liegt. Im Programm wird die Einschließung $\langle \pi/2 \rangle$ von $\pi/2$ realisiert durch die staggered Variable `pih_sm`. Weitere Einzelheiten zum Algorithmus findet man in dem nachfolgenden, leicht lesbaren **PASCAL-XSC** Quelltext.

2.3.0.4.11 Der Programm-Quelltext

```
function arg(var x,y : Stagg) : Stagg[-1..Stagg_prec];
{ Einschliessung von arg(x,y); x,y sind Koordin.-Intervalle }
{ (0 in x) and (0 in y) then error with program stop.       }
var t,u,v       : Stagg[-1..Stagg_prec];
   xi           : interval;
   neg,niu,niv  : boolean;
   r            : real;
begin
   v := y;  u := x;  Normal(u);  Normal(v);
   niu := 0 in u;   niv := 0 in v;
   if niu and niv then s_fu1_math_error(16);
   xi := arg(ival(u),ival(v));
   if xi = 0 then t := 0 else
   if diam(xi)/sup( abs(xi) ) > 1e-11 then t := xi
                                       else
   if niu then
   begin { 0 in u, aber 0 nicht in v enthalten }
      r := v[1]+v[-1];
      if (u[1]+u[0] >= r) or (-u[1]-u[-1] >= r)
         then t := arg(ival(u),ival(v)) else
         begin
            neg := v < 0;
            if neg then v := -v;
```

```
            t := inf(v);
            v := pih_sm - arctan(sup(u)/t);
            t := pih_sm - arctan(inf(u)/t);
            t := hull(v,t);
            if neg then t := -t;
         end;
   end    else
   { 0 nicht in u enthalten, aber 0 in v ist moeglich }
   if SIGN(u[1]+u[-1]) > 0 then
   begin                         { Rechter Halbraum: }
      neg := v < 0;  if neg then v := -v;
      if niv then
         if v[0]->v[-1] > u[1]+u[-1]
            then t := arg(ival(u),ival(v))
            else t := arctan(y/x)
             else
      if u[1]+u[-1]>v[1]+v[0] then t := arctan(v/u)
                              else t := pih_sm - arctan(u/v);
      if neg then t := -t;
   end                     else  { Linker Halbraum: }
   if niv then { 0 in y=v enthalten: }
      if (SIGN(v[1]+v[0])=0) and (SIGN(v[1]+v[-1])<0)
         then t := -pi_sm + arctan(v/u)
         else t :=  pi_sm + arctan(v/u)
          else  { 0 nicht in y=v enthalten: }
      if SIGN(v[1]+v[-1]) > 0 then { 3. Quadrant: }
         if v[1]+v[-1] >= -u[1]-u[-1]
            then t := pih_sm - arctan(u/v)
            else t := pi_sm + arctan(v/u)
                              else { 4. Quadrant: }
         if v[1]+v[0] < u[1]+u[-1]
            then t := -pih_sm - arctan(u/v)
            else t := -pi_sm + arctan(v/u);
   arg := t;
end; { arg }
```

Beachten Sie bitte, daß man mit obiger Funktion `arg(x,y)` aus dem Modul `s_fu1` den Imaginärteil der komplexen Logarithmus–Funktion im staggered Format realisieren kann. Im Bedarfsfall wird dabei $\ln(z)$ über die negative reelle Achse hinaus in den 4. Quadranten analytisch fortgesetzt.

2.3.0.4.12 Numerische Ergebnisse

Das **PASCAL-XSC** Programm

```
program tests;
use type_def, stagg, s_fu1;
var p : integer;
 procedure main;
 var x,y,z : Stagg[-1..Stagg_prec];
 begin
    repeat
       writeln('x: Stagg = ?');  read(x);
       writeln('y: Stagg = ?');  read(y);
       z := arg(x,y);
       writeln('arg(x,y) enthalten in: ');
       write(z);  writeln;
    until false;
 end; { main }
begin
   writeln('Stagg_prec = ?');   read(p);
   set_Stagg_prec(p);  main;
end.
```

liefert für den Wertebereich $\arg(x, y)$ die folgenden Einschließungen:

```
Stagg_prec = ?
5      <enter>
x : Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 [-2,-1]  <enter>
y : Stagg = ?
5 Stagg components and 1 interval are needed in this sequence
0 0 0 0 0 0        <enter>
arg(x,y) enthalten in:
 3.141592653589E+000
 7.932384626433E-013
 8.327950288419E-026
 7.169399375105E-039
 8.209749445923E-052
[  7.816406286208E-066,  7.816406286209E-066 ]
```

```
Stagg_prec = ?
2
x : Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-1e255,-1]
y : Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-1e-267,0]
arg(x,y) enthalten in:
-3.141592653589E+000
-7.932384626433E-013
[ -8.327950288420E-026, -8.327950288419E-026 ]

Stagg_prec = ?
2      <enter>
x : Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-1e255,-1]          <enter>
y : Stagg = ?
2 Stagg components and 1 interval are needed in this sequence
0 0 [-1e-267,+1e-267]    <enter>
arg(x,y) enthalten in:
 3.141592653589E+000
 7.932384626433E-013
[  8.327950288419E-026,  8.327950288420E-026 ]

Stagg_prec = ?
3      <enter>
x : Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
-1 0 0 0    <enter>
y : Stagg = ?
3 Stagg components and 1 interval are needed in this sequence
-1 0 0 0    <enter>
arg(x,y) enthalten in:
-2.356194490192E+000
-3.449288469825E-013
-3.745962716314E-026
[ -7.877049531330E-039, -7.877049531329E-039 ]
```

```
Stagg_prec = ?
18     <enter>
x : Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
y : Stagg = ?
18 Stagg components and 1 interval are needed in this sequence
2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  <enter>
arg(x,y) enthalten in:
 1.107148717794E+000
 9.050301706546E-014
 1.785370400700E-028
 4.764540143264E-041
 6.676539207433E-054
 7.103389773627E-067
 9.401341712868E-080
 6.170641434544E-093
 1.910054503158E-106
 1.004110412315E-119
 2.799603911491E-133
 3.412013493800E-146
 5.805785186089E-159
 1.590202770663E-172
 2.354867194833E-185
 7.093046927250E-198
 5.464279291462E-211
 2.530691740937E-224
[  7.626797415839E-237,  7.626797415840E-237 ]

Stagg_prec = ?
1      <enter>
x : Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 [1e-267,9e255]   <enter>
y : Stagg = ?
1 Stagg components and 1 interval are needed in this sequence
0 0   <enter>
arg(x,y) enthalten in:
 0.000000000000E+000
[ -0.000000000000E+000,  0.000000000000E+000 ]
```

```
Stagg_prec = ?
10     <enter>
x : Stagg = ?
10 Stagg components and 1 interval are needed in this sequence
1e50 0 0 0 0 0 0 0 0 0 0  <enter>
y : Stagg = ?
10 Stagg components and 1 interval are needed in this sequence
-1 0 0 0 0 0 0 0 0 0 0    <enter>
arg(x,y) enthalten in:
-9.999999999999E-051
-9.999999999999E-064
-9.999999999999E-077
-9.999999999999E-090
-9.999999999999E-103
-9.999999999999E-116
-9.999999999999E-129
-9.999999996666E-142
-6.666666666666E-155
-6.666666666666E-168
[ -6.666666666667E-181, -6.666666666666E-181 ]

Stagg_prec = ?
9      <enter>
x : Stagg = ?
9 Stagg components and 1 interval are needed in this sequence
1e-200 0 0 0 0 0 0 0 0 0  <enter>
y : Stagg = ?
9 Stagg components and 1 interval are needed in this sequence
-1e255 0 0 0 0 0 0 0 0 0  <enter>
arg(x,y) enthalten in:
-1.570796326794E+000
-8.966192313216E-013
-9.163975144209E-026
-8.584699687552E-039
-9.104874722961E-052
-5.390820314310E-065
-4.499314017412E-078
-6.710585339910E-091
-7.404325664115E-104
[ -3.323546922305E-117, -3.323546922304E-117 ]
```

Literaturverzeichnis

[1] M. Abramowitz and I.A. Stegun, editors. *Handbook of Mathematical Functions, Graphs, and Mathematical Tables*. Dover Publikations, INC., NEW YORK, 1972. ISBN 0-486-61272-4.

[2] G. Alefeld and J. Herzberger. *Einführung in die Intervallrechnung*. Reihe Informatik 12. **BI** Bibliographisches Institut Mannheim Wien Zürich, 1974. ISBN 3-411-01466-0.

[3] Konrad Knopp. *Theorie und Anwendung der unendlichen Reihen*. Grundlehren der mathematischen Wissenschaften Band 2. Springer-Verlag, Berlin Heidelberg New York, 1964. ISBN 3-540-03138-3.

[4] W. Krämer, U. Kulisch, and R. Lohner. Numerical Toolbox for Verified Computing II. Veröffentlichung geplant im Springer Verlag.