stichting
mathematisch
centrum

MC

AFDELING NUMERIEKE WISKUNDE NW NW 7/73 OCTOBER

P.W. HEMKER, W. HOFFMANN, S.P.N. VAN KAMPEN,
H.L. OUDSHOORN and D.T. WINTER

SINGLE AND DOUBLE-LENGTH COMPUTATION OF ELEMENTARY FUNCTIONS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

AMS(MOS) subject classification scheme (1970): 65D15, 65D20

CONTENTS

Abstract

A description in ALGOL 60 is given of procedures for the single and double-length computation of the following elementary functions: sqrt, exp, ln, sin, cos, tan, arcsin, arccos, arctan.
The algorithms for the single length computation of the ALGOL 60 reference language standard functions are equivalent to the algorithms used in the MC-ALGOL 60 system for the Electrologica X8 computer.
The double-length computation is accomplished with the aid of a floating point technique for extending the available precision as published by T.J. Dekker [1971].

Preface

The purpose of this report is twofold. Firstly, it describes in ALGOL 60 the routines for the standard functions as they are used in the MC-ALGOL 60-system for the Electrologica X8 computer. The ELAN source text for this system was written by Kruseman Aretz and Mailloux [1966]. The algorithm for the computation of the sine and the cosine was newly designed by Kruseman Aretz. An earlier version of the other algorithms was published by Barning [1965].
The series of ALGOL 60 procedures has been extended with a number of elementary functions which do not belong to the set of ALGOL 60 reference language standard functions. Moreover, a new algorithm for the computation of the natural logarithm has been inserted.

Secondly, this report describes a series of ALGOL 60 procedures which compute the elementary functions with double precision. These procedures use the double-length representation of floating-point numbers as proposed by Dekker [1972]. Also two methods are described for the normalization of these double-length floating-point numbers which enable us to define relational operators.

In order to read the double-length numbers into or out of the computer in decimal notation, it was necessary to construct a number of input/output routines. These routines are described in chapter 3.

Authors

The responsability for the different parts of this report was carried by

P.W. Hemker (ed.)	sections: 0.3, 1.4, 3.3, 4.4.
W. Hoffmann	sections: 1.2, 1.3, 4.2.
S.P.N. van Kampen	sections: 0.1, 1.7, 4.6, 4.7.
H.L. Oudshoorn	sections: 0.2, 1.1.
D.T. Winter	sections: 1.3, 1.5, 1.6, 2.1, 2.2, 2.3, 3.1, 3.2, 4.1, 4.3, 4.5.

0. Preliminary notes

0.1. General remarks on polynomial approximation

In this section we give a short description of some well-known methods used for the polynomial approximations of the elementary functions in chapter 1 and 4.
First we describe the method of truncation of a Taylorseries and next the method to get a more efficient approximation by means of Chebyshev polynomials or by a minimax approximation.
For a more detailed treatment of Chebyshev polynomials the reader is referred to e.g. Lanzos [1956] and Fike [1968] and for the minimax approximation to Fike [1968] and Meinardus [1967]. A more theoretical view is given by Achieser [1953] and also by Meinardus [1967].

When we approximate a function, there are mainly two sources of errors. First the rounding errors which arise from the finite precision with which each operation is performed. The effect of rounding errors depends strongly on the particular method of computation and is treated for each function separately in chapter 1. Second we have to deal with the truncation error, arising from the approximation of an infinite process by a finite process. This truncation error must be small with respect to the rounding error.

Because it is possible by means of the transformation

$$z = \frac{2x-(b+a)}{b-a} \tag{0.1.1}$$

to change the interval [a,b] into [-1,1], we will confine ourselves to approximations in the interval [-1,1].

Truncation of Taylorseries

Let

$$f(x) = \sum_{k=0}^{\infty} a_k x^k$$

be a convergent Taylorseries of the function $f(x)$ on the interval $[-1,1]$. If we truncate this series after n terms we obtain a polynomial $p_n(x) = a_0+a_1x+\ldots+a_nx^n$ of degree $\leq n$. $p_n(x)$ is an approximation to $f(x)$. As a bound for the magnitude of the absolute truncation error in the range $[-1,1]$ we find

$$|f(x)-p_n(x)| \leq |a_{n+1}| + |a_{n+2}| + \ldots \quad (0.1.2)$$

If $|f(x)| \geq m > 0$ holds in the approximation interval $[-1,1]$, a bound for the relative truncation error is given by

$$\left|\frac{f(x)-p_n(x)}{f(x)}\right| \leq \frac{1}{m}\left(|a_{n+1}| + |a_{n+2}| + \ldots\right).$$

<u>Example 1</u>

When we approximate $\sin(x\pi/4)$ by a truncated power series in $[-1,1]$, it is not possible to give a bound for the relative error because the function $\sin(x\pi/4)$ has a zero for $x = 0$. We avoid this by approximating $\frac{\sin(x\pi/4)}{x}$. The Taylorseries of this function reads

$$\frac{\sin(x\pi/4)}{x} = \sum_{k=0}^{\infty} (-1)^k \frac{(\pi/4)^{2k+1}}{(2k+1)!} x^{2k}.$$

Suppose we want a truncation error less than $0.5_{10}{-12}$.
Truncating this series after the seventh term we obtain a polynomial of degree 12:

$$p_{12}(x) = \sum_{k=0}^{6} (-1)^k \frac{(\pi/4)^{2k+1}}{(2k+1)!} x^{2k}.$$

Using this polynomial as an approximation to $\frac{\sin(x\pi/4)}{x}$ on $[-1,1]$, we get the following bound for the absolute error

$$\left|\frac{\sin(x\pi/4)}{x} - p_{12}(x)\right| \leq \frac{(\pi/4)^{15}}{15!} \approx 0.212_{10}{-13}.$$

Since $\left|\frac{\sin(x\pi/4)}{x}\right| \geq \frac{1}{2}\sqrt{2}$ on [-1,1], a bound for the relative error is

$$\left|\frac{\frac{\sin(x\pi/4)}{x} - p_{12}(x)}{\frac{\sin(x\pi/4)}{x}}\right| \leq \sqrt{2}\,\frac{(\pi/4)^{15}}{15} \approx 0.299_{10}{-13}.$$

Approximating $\sin(x\pi/4)$ by $xp_{12}(x)$, we find a polynomial of degree 13, of which the coefficients of the even terms are zero and with the same bounds for the absolute and relative truncation error.

Economization with Chebyshev polynomials

A more efficient approximation to a function $f(x)$ is possible by means of the method of economization (or telescoping) of the truncated Taylor-series with Chebyshev polynomials.

Chebyshev polynomials form a class of orthogonal polynomials in the interval [-1,1] and are defined by

$$T_n(x) = \cos(n \arccos x), \quad n=0,1,2,\ldots \;. \tag{0.1.3}$$

From this definition we may derive the recursion formula

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{0.1.4}$$

Starting with $T_0(x) = 1$ and $T_1(x) = x$ we find

$$\begin{aligned} T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x, \\ T_4(x) &= 8x^4 - 8x^2 + 1, \\ T_5(x) &= 16x^5 - 20x^3 + 5x, \\ &\ldots\ldots\ldots\ldots\ldots \;. \end{aligned}$$

It is easy to verify that $T_n(x)$ is a polynomial of degree n with leading coefficient 2^{n-1} for n=1,2,..., while $T_n(x)$ is an even function if n is even and an odd function if n is odd.

From the defining formula it appears that $T_n(x_k) = 0$ for $x_k = \cos((2k-1)\pi/2n)$, k=1,2,...,n, and $-1 \le T_n(x) \le 1$, while $T_n(x_k) = \pm 1$ for $x_k = \cos(k\pi/n)$, k=0,1,...,n. Note that $T_n(x)$ has n real and distinct zeros in [-1,1] and n+1 equal extreme values with alternating sign in the range [-1,1].

By means of the inverse of the transformation (0.1.1) we obtain shifted Chebyshev polynomials $T_n^{[a,b]}(z)$ with the same properties in the interval [a,b] as $T_n(x)$ in the interval [-1,1]. Generally the transformation of $T_n(x)$ to $T_n^{[a,b]}(z)$ is a more stable process than the transformation of the function on [a,b] to the shifted function on [-1,1].

An important property of Chebyshev polynomials is the minimax property: From all polynomials $p_n(x)$ of degree n with leading coefficient 1, $2^{1-n}\ T_n(x)$ has a minimal maximum value 2^{1-n} in the interval [-1,1]. That is

$$\max_{x\in[-1,1]} |p_n(x)| \ge 2^{1-n}$$

and

$$\max_{x\in[-1,1]} |p_n(x)| = 2^{1-n}$$

if and only if $p_n(x) = 2^{1-n}\ T_n(x)$.
For a proof of this property see e.g. Achieser [1953] or Fike [1968].

Let $p_n(x) = a_0+a_1x+\ldots+a_nx^n$ be an approximation to f(x) on [-1,1] obtained by truncation of the Taylorseries. We have seen that (0.1.2) gives a bound for the absolute truncation error of this approximation. We now define a polynomial $p_{n-1}(x)$ of degree $\le$ n-1 as follows

$$p_{n-1}(x) = p_n(x) - a_n\ 2^{1-n}\ T_n(x).$$

Because $(p_n(x)-p_{n-1}(x))/a_n$ is a polynomial of degree n with leading coefficient 1, we have

$$\max_{x\in[-1,1]} |(p_n(x)-p_{n-1}(x))/a_n| = \max_{x\in[-1,1]} |2^{1-n}\, T_n(x)| = 2^{1-n} ,$$

and because of the minimax property of the Chebyshev polynomials, $p_{n-1}(x)$ is the optimal approximation to $p_n(x)$ on [-1,1] with a bound for the absolute truncation error

$$|p_n(x) - p_{n-1}(x)| \le |a_n|\, 2^{1-n} .$$

If f(x) is an even function, $p_n(x)$ and $T_n(x)$ will be even functions and $p_{n-1}(x)$ will be of degree ≤ n-2. For an odd function f(x) we have similar properties for $p_{n-1}(x)$.

Considering $p_{n-1}(x)$ as an approximation to f(x) we find a bound for the magnitude of the absolute truncation error on [-1,1]

$$\begin{aligned} |f(x) - p_{n-1}(x)| &\le |p_n(x) - p_{n-1}(x)| + |f(x) - p_n(x)| \\ &\le |a_n|\, 2^{1-n} + |a_{n+1}| + |a_{n+2}| + \dots . \end{aligned}$$

By repeating the process of economization on the polynomial $p_{n-1}(x)$ we obtain an approximation $p_{n-2}(x)$ to f(x) with as a bound for the absolute truncation error

$$|f(x) - p_{n-2}(x)| \le |a'_{n-1}|2^{2-n}+|a_n|2^{1-n}+|a_{n+1}|+|a_{n+2}|+ \dots ,$$

with a'_{n-1} the leading coefficient of $p_{n-1}(x)$.

<u>Example 2</u>.

We want to approximate arctan x in the range [-tan(π/12), tan(π/12)] with an absolute truncation error less than $0.5_{10}{}^{-12}$ (see section 1.7). In this range we have a convergent Taylorseries

$$\arctan x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{2k+1} . \qquad (0.1.5)$$

Because we want to keep the first coefficient equal to one, we consider the series

$$\frac{\dfrac{\arctan x}{x} - 1}{x^2} = \sum_{k=1}^{\infty} (-1)^k \frac{x^{2k-2}}{2k+1} \quad .$$

Putting now $x = z\tan(\pi/12)$ we obtain

$$f(z) = \frac{\dfrac{\arctan(z\tan(\pi/12))}{z\tan(\pi/12)} - 1}{z^2(\tan(\pi/12))^2} = \sum_{k=1}^{\infty} (-1)^k \frac{(\tan(\pi/12))^{2k-2}}{2k+1} z^{2k-2},$$

with z in the range [-1,1]. By truncating this series after the ninth term we obtain the polynomial approximation to f(z)

$$p_{16}(z) = a_0 + a_2 z^2 + \ldots + a_{16} z^{16}$$

with

$$a_{2k} = (-1)^{k+1} \frac{(\tan(\pi/12))^{2k}}{2k+3} \quad , \qquad k=0,1,\ldots,8 \ ,$$

and as a bound for the truncation error

$$|f(z) - p_{16}(z)| \le |a_{18}| = (\tan(\pi/12))^{18}/21 \approx 0.242_{10}{-11} \ .$$

By using $x + x^3 p_{16}(x)$ with $x=z\tan(\pi/12)$ as an approximation to arctan x we obtain the truncation error

$$|\arctan x - (x + x^3 p_{16}(x))| \le |a_{18}|(\tan(\pi/12))^3 \approx 0.464_{10}{-13} \ .$$

Economizing $p_{16}(z)$, we obtain an approximation of degree 14

$$p_{14}(z) = p_{16}(z) - a_{16} 2^{-15}\, T_{16}(z) \ ,$$

with the following bound for the truncation error

$$|f(z) - p_{14}(z)| \leq |a_{16}|2^{-15} + |a_{18}| \approx 0.242_{10}{}^{-11} \; .$$

For the approximation to arctan x the bound for the truncation error is

$$|\arctan x - (x + x^3 p_{14}(x))| \leq (|a_{16}|2^{-15} + |a_{18}|)\;(\tan(\pi/12))^3 \approx$$

$$\approx 0.465_{10}{}^{-13} \; .$$

Repeating the economization process on $p_{14}(z)$ and $p_{12}(z)$ we obtain a polynomial with a bound for the absolute truncation error

$$|f(z) - p_{10}(z)| \leq |a'_{12}|2^{-11} + |a'_{14}|2^{-13} + |a_{16}|2^{-15} + |a_{18}| \approx 0.608_{10}{}^{-11} .$$

Putting $x=z \tan(\pi/12)$ we obtain as an approximation to arctan x a polynomial of degree 13 with odd terms only

$$\begin{aligned} \arctan x = x &- 0.33333\;33333\;297x^3 + 0.19999\;99963\;100x^5 \\ &- 0.14285\;65386\;578x^7 + 0.11107\;47892\;853x^9 \qquad (0.1.6) \\ &- 0.08991\;43448\;436x^{11} + 0.06411\;78597\;458x^{13} + \varepsilon \end{aligned}$$

with $|\varepsilon| < 0.608_{10}{}^{-11}\;(\tan(\pi/12))^3 \approx 0.117_{10}{}^{-12}$.

If we economize $p_{10}(z)$ once more we obtain $p_8(z)$ with

$$\max_x |\arctan x - (x + x^3 p_8(x))| > 0.5_{10}{}^{-12} \; .$$

If we start the economization process with a truncated Taylorseries of degree 14 instead of 16, we would have found as a bound for the truncation error $0.783_{10}{}^{-12} > 0.5_{10}{}^{-12}$.

Minimax approximation

Although the method of economization gives a good approximation of a function, it is possible to obtain an optimal polynomial approximation of degree n. A polynomial $p_n(x)$ of degree n is an optimal approximation to

$f(x)$ in the range $[a,b]$ with respect to the absolute error, if $\max_{x\in[a,b]} |f(x)-p_n(x)|$ is minimal. In this case $p_n(x)$ is called a minimax approximation to $f(x)$ on $[a,b]$.

An important theorem on minimax polynomial approximations is the theorem of P.L. Chebyshev:

Let $f(x)$ be a function continuous on $[a,b]$ and let P_n denote the set of all polynomials of degree $\le n$. Then there exists a unique polynomial p_n^* in P_n such that

$$\max_{x\in[a,b]} |f(x)-p_n^*(x)| = \min_{p_n\in P_n} \max_{x\in[a,b]} |f(x)-p_n(x)| .$$

If $p_n \in P_n$ then $p_n \equiv p_n^*$ if and only if there exist $N \ge n+2$ points

$$a \le x_1 < x_2 < \ldots < x_N \le b$$

such that

$$f(x_k) - p_n(x_k) = (-1)^k\, m, \quad k=1,2,\ldots,N,$$

where $m = \max_{x\in[a,b]} |f(x)-p_n(x)|$.

For a proof of this theorem see e.g. Meinardus [1967] or Achieser [1953].

<u>Example 3</u>.

Let $f(x) = a_0 + a_1 x + \ldots + a_n x^n$ be a function with $a_n \ne 0$. We define an approximation to $f(x)$ of degree $\le n-1$ on $[-1,1]$ by means of

$$p_{n-1}(x) = f(x) - a_n 2^{1-n}\, T_n(x) ,$$

with error function

$$\varepsilon(x) = f(x) - p_{n-1}(x) = a_n 2^{1-n}\, T_n(x).$$

$\varepsilon(x)$ has extreme values $(-1)^k a_n 2^{1-n}$ at n+1 points $x_k = \cos(k\pi/n)$, $k = 0,1,\ldots,n$. It follows from the theorem of Chebyshev that $p_{n-1}(x)$ is the unique minimax approximation to f(x) with degree ≤ n-1 on [-1,1]. Note that

$$p_{n-2}(x) = p_{n-1}(x) - a'_{n-1} 2^{2-n}\, T_{n-1}(x)$$

is not the minimax approximation of degree ≤ n-2 to f(x).

There exist several methods to obtain a minimax polynomial approximation. Here we give a short description of the iterative method of E.Ja. Remez [1934].

Let $p_n(x) = a_0 + a_1 x + \ldots + a_n x^n$ be the minimax approximation to f(x) on [a,b]. According to the theorem of Chebyshev there exist n+2 points

$$a \le x_1 < x_2 < \ldots < x_{n+2} \le b\ ,$$

for which $f(x) - p_n(x)$ has extreme values $(-1)^k m$. If the critical points x_k were known, we could solve the linear equations

$$a_0 + a_1 x_k + \ldots + a_n x_k^n + (-1)^k m = f(x_k), \quad k=1,2,\ldots,n+2\ ,$$

with n+2 unknown variables $a_0, a_1, \ldots, a_n$, m. The critical points x_k can be approximated by the following iterative algorithm:

step 1: Initially take for the critical points

$$x_k = \tfrac{1}{2}(b-a)\cos((n-k+2)\pi/(n+1)) + \tfrac{1}{2}(b+a), \; k=1,2,\ldots,n+2\ ,$$

according to the critical values of the shifted Chebyshev polynomial $T_{n+1}^{[a,b]}(x)$.

step 2: Compute the n+2 variables $a_0^*, a_1^*, \ldots, a_n^*$, m^* by solving the linear system of n+2 equations

$$a_0^* + a_1^* x_k + \ldots + a_n^* x_k^n + (-1)^k m^* = f(x_k), \quad k=1,2,\ldots,n+2.$$

$a_0^*, a_1^*, \ldots, a_n^*$ are approximations to the coefficients $a_0, a_1, \ldots, a_n$ of $p_n(x)$.

step 3: If $\max_{x\in[a,b]} |f(x)-(a_0^*+a_1^*x+\ldots+a_n^*x^n)|$ exceeds a chosen tolerance, then compute n+2 extreme points for which $f(x) - (a_0^*+a_1^*x+\ldots a_n^*x^n)$ attains extreme values and go to step 2.

For a proof of the convergence of the method see Meinardus [1967].

Example 4.

We want to approximate arctan x in the range $[-\tan(\pi/12), \tan(\pi/12)]$ by a minimax polynomial approximation with an absolute truncation error less than $0.5_{10}{-12}$. In the approximation we want to have the first coefficient equal to one and the coefficients of the even terms zero. Therefore we substitute $y=x^2$ and write the Taylorseries (0.1.5) in the form

$$\frac{\dfrac{\arctan\sqrt{y}}{\sqrt{y}} - 1}{y} = \sum_{k=1}^{\infty} (-1)^k \frac{y^{k-1}}{2k+1} . \qquad (0.1.7)$$

It is possible to compute a minimax approximation to

$$\sum_{k=1}^{\infty} (-1)^k \frac{y^{k-1}}{2k+1} ,$$

but as a consequence of the effect of rounding errors during the computation this will not give an optimal result. Therefore we write (0.1.7) in the form

$$\frac{\dfrac{\arctan\sqrt{y}}{\sqrt{y}} - 1}{y} - \sum_{k=1}^{5} (-1)^k \frac{y^{k-1}}{2k+1} = \sum_{k=6}^{\infty} (-1)^k \frac{y^{k-1}}{2k+1} .$$

Applying the Remez' algorithm to

$$\sum_{k=6}^{14} (-1)^k \frac{y^{k-1}}{2k+1}$$

on $[0, (\tan(\pi/12))^2]$, we obtain a minimax approximation

$$
\begin{aligned}
p_5(y) = {} & 0.35924\,61635\,246_{10}{}^{-11} - 0.36404\,32526\,113_{10}{}^{-8}y \\
& + 0.59828\,58022\,561_{10}{}^{-6}y^2 - 0.36070\,65345\,046_{10}{}^{-4}y^3 \\
& + 0.99028\,36679\,849_{10}{}^{-3}y^4 - 0.12776\,43575\,806_{10}{}^{-1}y^5 + \varepsilon
\end{aligned}
$$

with $|\varepsilon| < 0.360_{10}{}^{-11}$. For arctan x we obtain the approximation

$$\arctan x = x + x^3 \sum_{k=1}^{5} (-1)^k \frac{x^{2k-2}}{2k+1} + x^3 p_5(x^2) + x^3\varepsilon$$

which results in

$$
\begin{aligned}
\arctan x = x & - 0.33333\,33333\,298x^3 + 0.19999\,99963\,596x^5 \\
& - 0.14285\,65445\,713x^7 + 0.11107\,50404\,576x^9 \qquad (0.18) \\
& - 0.08991\,88072\,411x^{11} + 0.06414\,66411\,651x^{13} + \varepsilon'
\end{aligned}
$$

with $|\varepsilon'| < 0.691_{10}{}^{-13}$.

Approximating arctan x by the minimax polynomial of degree 11, we obtain as a bound for the truncation error $0.460_{10}{}^{-11} > 0.5_{10}{}^{-12}$.

Comparing the approximations (0.1.6) and (0.1.8) of example 2 and 4, respectively, we see that, although the minimax approximation has the smaller truncation error, this does not result in a shorter polynomial approximation. As the economization method gives a near-minimax approximation, this will be the same for most approximations to a function in a small range. Hence, for optimal polynomial approximations we can use economization, which is easier to apply than the minimax method.

0.2. The computer arithmetic of the EL X8

The EL X8 is a binary computer with a word length of 27 bits.
Fixed-point numbers (integers) are represented according to the One Complement System. The range for integers is [-67108863, + 67108863]. Floating-point numbers have the Grau-representation [Grau, 1962], i.e. $x = mx\ \beta^{ex}$ where for the EL X8:

$$|mx| \leq 2^{40} - 1 , \quad \beta = 2 \text{ and } |ex| \leq 2^{11} - 1.$$

So the smallest representable positive floating-point number (the dwarf) is 2^{-2047} ($\approx 10^{-616}$) and the largest (the giant) is $(2^{40}-1) * 2^{2047}$ ($\approx 10^{628}$).

When, due to an arithmetic operation, overflow occurs, the giant is delivered as the result; in case of underflow the dwarf, in both cases with the correct sign.
A result equal to zero only is obtained in the following cases:

a + b = 0 iff a and -b have the same bit pattern,
a - b = 0 iff a and b have the same bit patter ,
a * b = 0 iff a = 0 or b = 0,
a / b = 0 iff a = 0.

The floating-point arithmetic of the EL X8 is optimal, i.e. when no over- or underflow occurs the result delivered is the representable number nearest to the exact result. In case of ambiguity the in absolute value largest of the two possible numbers is delivered; only when an addition of two numbers with opposite sign or a subtraction of two numbers with equal sign is concerned the in absolute value smallest number is delivered.

0.3. Rounding errors in polynomial approximations

In section 0.1 the influence of the truncation error was stressed. However, when polynomial approximations are required which are accurate up to the machine precision, other sources of error also have to be considered. Firstly, the coefficients of the approximating polynomial can only be represented in finite precision. This implies that the polynomial used differs from the required approximating polynomial.
Secondly, rounding errors appear during the evaluation of the polynomial. It is difficult to give a systematic treatment of this type of error. The most important thing to do is to use a stable calculating process such that rounding errors will not be amplified in the remainder part of the computation. A stable algorithm will cause only the last few bits to be affected. In addition, it is sometimes possible to modify the coefficients of the approximating polynomial, in order to correct for the last marginal effects.
As an example we give in table 1 the coefficients of the sine- and cosine approximations obtained by economizing the powerseries (as described in section 0.1) and the modified coefficients, (corrected for the influence of their finite representation) as they are given in Kruseman Aretz and Mailloux [1966].

	coefficients econ. series	modified coefficients
c 0	+1	+1
c 2	$-.12337\ 00550\ 136_{10}{}^{+1}$	$-.12337\ 00550\ 125_{10}{}^{+1}$
c 4	$+.25366\ 95077\ 229$	$+.25366\ 95072\ 540$
c 6	$-.20863\ 47506\ 082_{10}{}^{-1}$	$-.20863\ 46891\ 369_{10}{}^{-1}$
c 8	$+.91919\ 63466\ 817_{10}{}^{-3}$	$+.91916\ 54179\ 155_{10}{}^{-3}$
c10	$-.24909\ 25342\ 519_{10}{}^{-4}$	$-.24856\ 34468\ 030_{10}{}^{-4}$
c 1	$+.15707\ 96326\ 794_{10}{}^{+1}$	$+.15707\ 96326\ 794_{10}{}^{+1}$
c 3	$-.64596\ 40975\ 045$	$-.64596\ 40974\ 927$
c 5	$+.79692\ 62615\ 575_{10}{}^{-1}$	$+.79692\ 62611\ 834_{10}{}^{-1}$
c 7	$-.46817\ 52591\ 814_{10}{}^{-2}$	$-.46817\ 52592\ 887_{10}{}^{-2}$
c 9	$+.16042\ 92666\ 630_{10}{}^{-3}$	$+.16042\ 92697\ 341_{10}{}^{-3}$
c11	$-.35563\ 88849\ 621_{10}{}^{-5}$	$-.35564\ 00770\ 321_{10}{}^{-5}$

Table 1.

c0 - c10 denote the coefficients of the cosine approximation.

c1 - c11 denote the coefficients of the sine approximation.

In order to illustrate the effects of both (1) the finite representation of the polynomial coefficients and (2) the rounding errors during the evaluation of the polynomial, we show in the figures 1 - 8

(1) the final truncation error, i.e.

$$P(2x/\pi) - \cos(2x/\pi),$$

where P denotes the approximating polynomial, and

(2) the final relative error obtained, i.e.

$$\frac{\cos_{\text{computed}}(2x/\pi) - \cos(2x/\pi)}{\cos(2x/\pi)} .$$

We computed these errors both for the coefficients of the economized power-series and for the modified coefficients. Similar results are given for the sine function.

ERRATUM

Report NW 7/73.
Page 20, lines 7 and 8 f.b. .

(1) the final truncation error, i.e.

$$P(2x/\pi) - \cos(2x/\pi),$$

should read:

(1) the final relative truncation error, i.e.

$$(P(2x/\pi) - \cos(2x/\pi))/ \cos(2x/\pi),$$

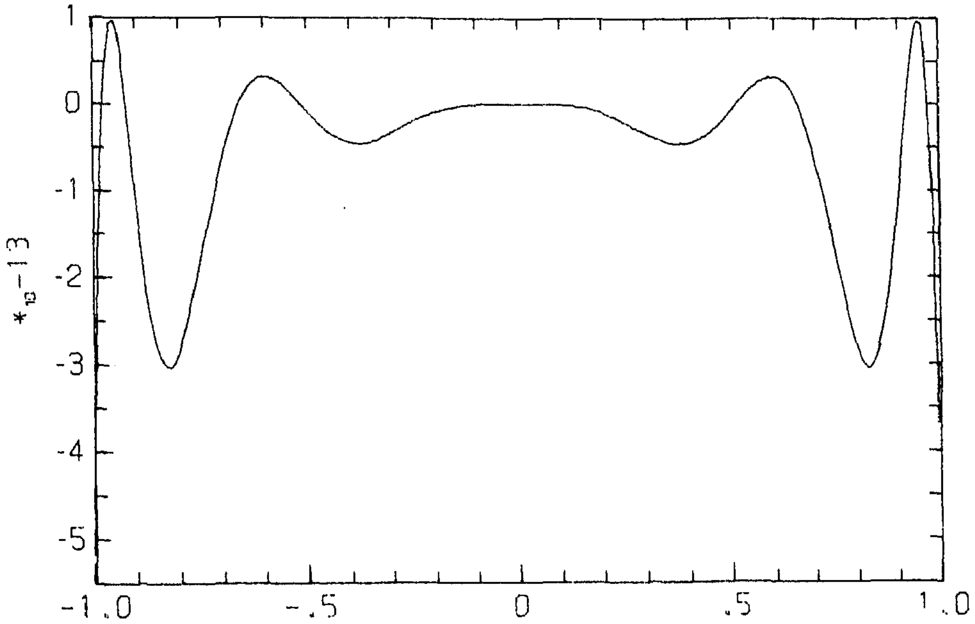

Fig.1. Truncation error, cosine function, coefficients economized powerseries.

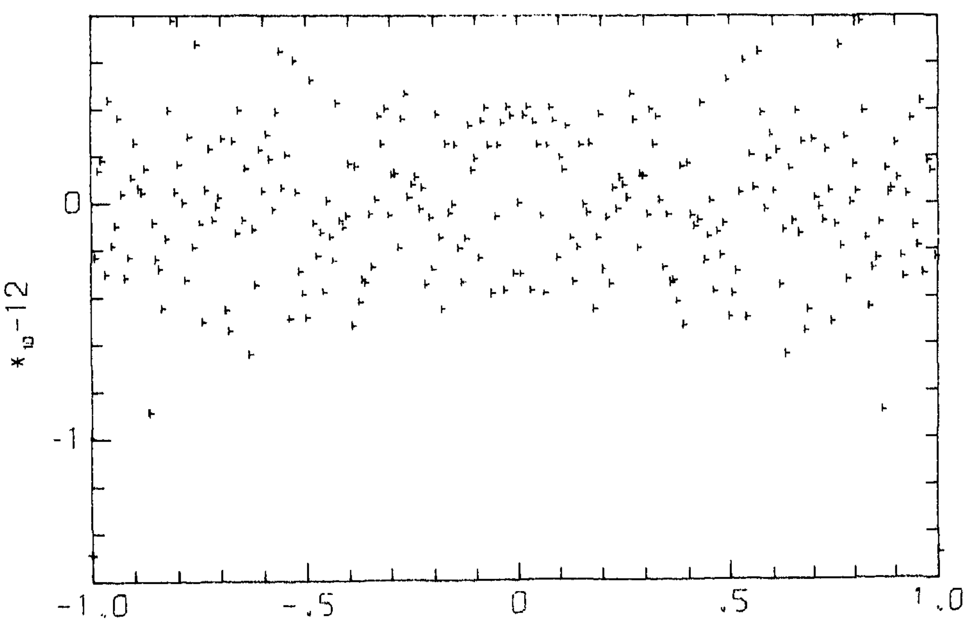

Fig.2. Relative error, cosine function, coefficients economized powerseries.

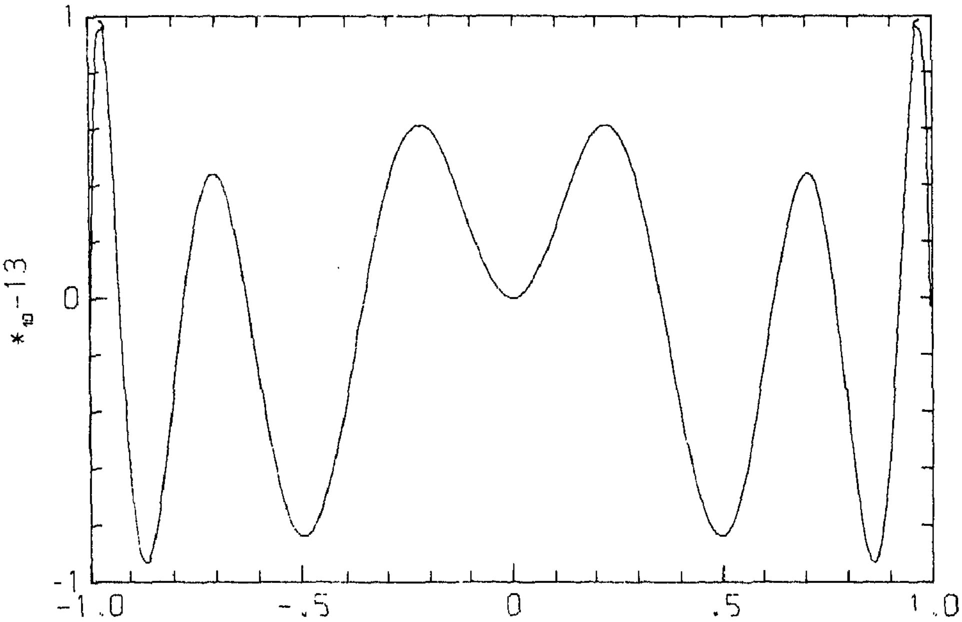

Fig.3. Trunction error, cosine function, modified coefficients.

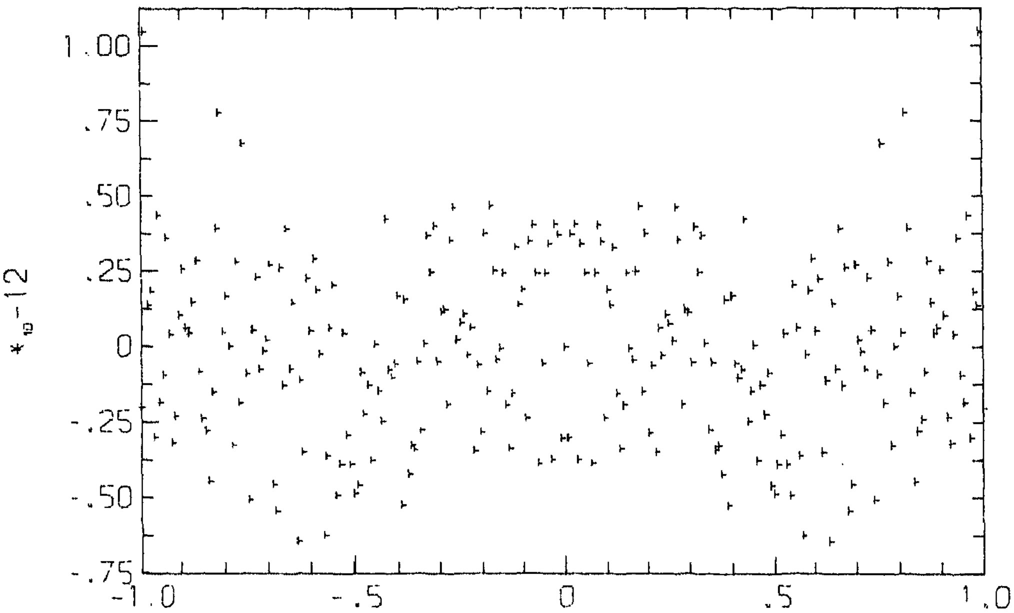

Fig.4. Relative error, cosine function, modified coefficients.

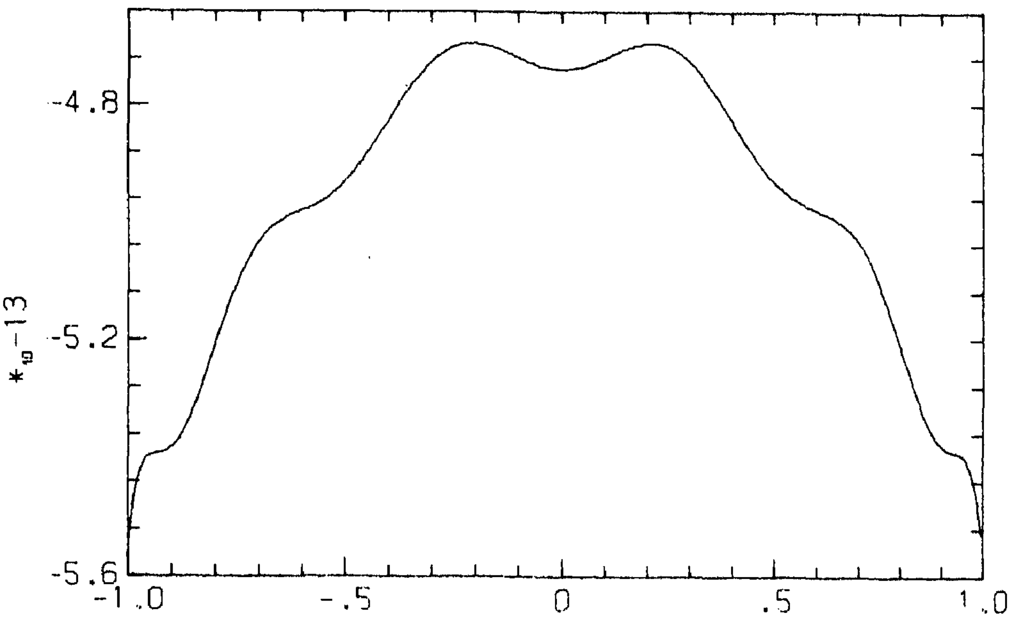

Fig.5. Truncation error, sine function,
coefficients economized powerseries.

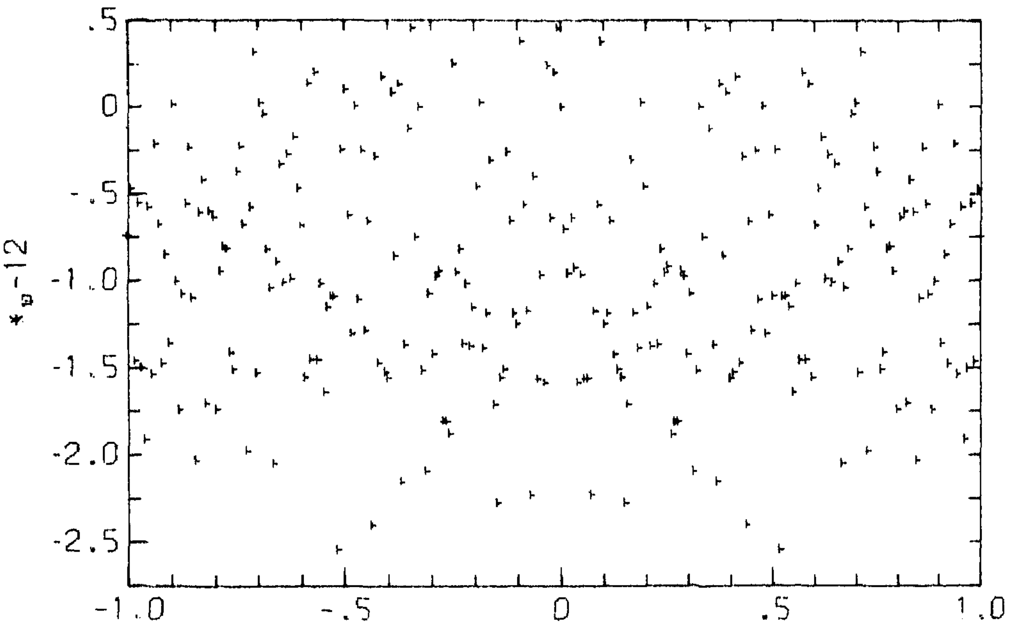

Fig.6. Relative error, sine function,
coefficients economized powerseries.

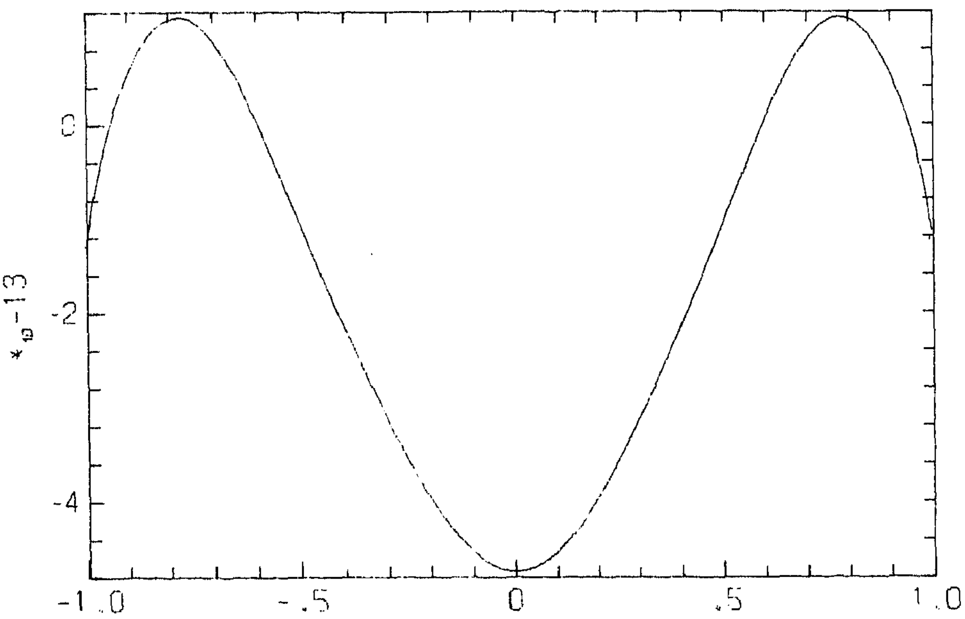

Fig.7. Truncation error, sine function, modified coefficients.

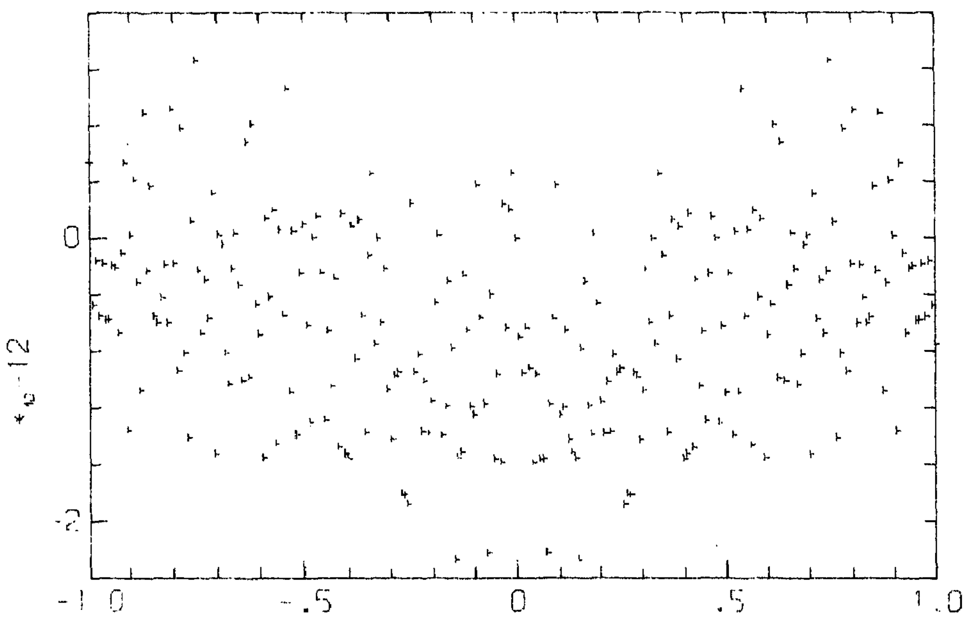

Fig.8. Relative error, sine function, modified coefficients.

1. The computation of single length elementary functions

1.1. Square root

The square root of a positive real number x is iteratively computed by the Newton-Raphson formula

$$x_{i+1} = (x_i + x/x_i)/2 \; . \qquad (1.1.1)$$

To guarantee the desired relative accuracy to be reached after a fixed number of iterations it is necessary to transform the positive real axis to a suitable small interval [a,b).

For the relative error $\delta_i(x) = \dfrac{x_i - \sqrt{x}}{\sqrt{x}}$ the following recursive relation holds:

$$\delta_{i+1} = \frac{\delta_i^2}{2(1+\delta_i)} \; . \qquad (1.1.2)$$

After a transformation $x \to x'$ ($x \in (0,\infty)$, $x' \in [a,b)$) the first order Chebyshev approximation can be taken as a starting value x_0, i.e. $x_0 = \alpha x' + \beta$, where α and β must be chosen such that $|\delta_0(a)| = |\delta_0(b)| = \max\limits_{a<\xi<b} |\delta_0(\xi)|$.

It is easily verified that:

1. $\beta = \alpha\sqrt{ab}$,
2. the point ξ' for which max $|\delta_0(\xi)|$ is achieved is $\xi' = \beta/\alpha = \sqrt{ab}$, (1.1.3)
3. $\alpha = 2/(\sqrt[4]{a} + \sqrt[4]{b})^2$.

The EL X8 standard routine sqrt(x) uses the interval $[\frac{1}{4},1)$. This reduction of the argument range is performed by multiplication by powers of two only, which does not introduce any error.
The values of α and β should be, according to (1.1.3),

$\alpha = .6862915010151$, $\beta = .3431457505076$.

It follows that:

$\delta_0(\frac{1}{4}) = \delta_0(1) = .2943725152_{10}-1$, $\delta_0(\xi') = -.2943725152_{10}-1$,

$\delta_1(\frac{1}{4}) = \delta_1(1) = .4208861570_{10}-3$, $\delta_1(\xi') = .4464171835_{10}-3$,

$\delta_2(\frac{1}{4}) = \delta_2(1) = .8853531527_{10}-7$, $\delta_2(\xi') = .9959968786_{10}-7$,

$\delta_3(\frac{1}{4}) = \delta_3(1) = .3919250678_{10}-14$, $\delta_3(\xi') = .4960048417_{10}-14$.

Aiming at a relative error of 2^{-40} two conclusions can be drawn:

(i) three iterations are necessary,

(ii) using the optimal values of α and β the relative error is much less than is required.

This justifies a choice of α and β, different from the optimal values. The EL X8 routine sqrt(x) uses the values $\alpha = 5/8$ (for efficient computation) and $\beta = .3656805753708 = 1 + \alpha/4 - \sqrt{\alpha}$ (optimal, given the constraint $\delta_0(\frac{1}{4}) = -\delta_0(\xi')$).
Now we obtain:

$\delta_3(\frac{1}{4}) = .8996199105_{10}-13$,

$\delta_3(\xi') = .1277836562_{10}-12$,

$\delta_3(1) = .4614558633_{10}-18$.

So the required relative accuracy will be reached.

Moreover, the result delivered by the procedure sqrt can be shown to be optimal. Let ε be the machine precision and assume the value of x_2 to be exact. Then x_3 is computed by

$$x_3 = fl(x_2+fl(x/x_2))/2 .$$

Keeping in mind the interval on which $\sqrt{x}$ is computed, we find the following for the upperbound of the absolute error:
the absolute error in $fl(x/x_2) \le \frac{1}{2}\varepsilon$,
the addition $fl(x_2+fl(x/x_2))$ gives an extra $\frac{1}{2}\varepsilon$, but division by 2, being exact, leads to an upperbound for the absolute error in x_3 of $\frac{1}{2}\varepsilon$.

As a consequence sqrt(x) exactly equals $\sqrt{x}$ whenever x and $\sqrt{x}$ can be represented exactly.

```
real procedure sqrt(x); value x; real x;
if x < 0 then sqrt:= 0 else
begin integer n, sgn; real x0;
     n:= bin exp(x, sgn);
     if (n : 2) × 2 ≠ n then begin n:= n + 1; x:= x/2 end;
     x0:= .625 × x + .3656805753708;
     x0:= (x/x0 + x0)/2;
     x0:= (x/x0 + x0)/2;
     x0:= (x/x0 + x0)/2;
     sqrt:= x0 × two ttp(n : 2)
end sqrt;

comment If overflow cannot occur (i.e. abs(int) < 2048)
        the procedure "two ttp" delivers 2↑int.
        Although an efficient procedure is only possible
        in machine-code, an equivalent ALGOL-version is
        given below;
real procedure two ttp(int); value int; integer int;
if int > 2047 then two ttp:= .1615850303566₁₀+617 else
if int < -2047 then two ttp:= .6188692094765₁₀-616 else
begin integer absint, n;
     real t, tt;
     absint:= abs(int); t:= 1; tt:= 2;
loop: n:= absint : 2;
     if n × 2 ≠ absint then t:= t × tt;
     absint:= n; if absint ≠ 0 then
     begin tt:= tt × tt; goto loop end;
     two ttp:= if int ≥ 0 then t else 1/t
end two ttp;

comment The procedure "bin exp" delivers the binary
        exponent of abs(x) as an integer value.
        Moreover, the sign of x is delivered in sgn
        and, if x ≠ 0, x is replaced by its binary
        mantissa (0.5 ≤ x < 1).
        Although an efficient procedure is only possible
        in machine-code, an equivalent ALGOL-version is
        given below;
integer procedure bin exp(x, sgn); real x; integer sgn;
begin integer i, e;
     sgn:= sign(x); x:= abs(x);
     if x = 0 then e:= 0 else
     if x < 1 then
     begin i:= e:= 0;
         for i:= i - 1 while x < 0.5 do
         begin e:= i; x:= x × 2 end
     end
     else
     for i:= 1, i + 1 while x ≥ 1 do
     begin e:= i; x:= x / 2 end;
     bin exp:= e
end bin exp;
```

1.2. Exponential function

The exponential function of a real number x: exp(x) is computed by a polynomial approximation of $2 \uparrow x$. For a good approximation it is necessary that the argument lies within a finite interval in which the function $2 \uparrow x$ is smooth enough.
Our choice for this interval is [-.5,0); hence the transformation is as follows.

Let

$$n = \text{entier } (x \times {}^2\log e) + 1$$

and

$$y = x \times {}^2\log e - n$$

then

$$\exp(x) = 2 \uparrow n \times 2 \uparrow y$$

with n integer and $y \in [-1,0)$.
If $y \in [-1,-.5)$ then we replace y by y/2 ;
and in this case we have $\exp(x) = 2 \uparrow n \times (2 \uparrow y) \uparrow 2$.
Now we approximate $2 \uparrow y$ on the interval $-.5 \le y < 0$.
We use the Taylorseries expansion:

$$2 \uparrow y = \sum_{k=0}^{\infty} \frac{(y \ln 2)^k}{k!} \quad .$$

With the telescoping technique (cf. section 0.1) we find a seventh degree polynomial

$$P(y) = \sum_{i=0}^{7} c_i y^i \text{ with } |P(y) - 2 \uparrow y| < .5_{10}{-12} \text{ if } y \in [-.5,0).$$

The coefficients of this polynomial are given in the procedure body.

To obtain the value of exp(x) we have to multiply the value of P(y) (or P(y) ↑ 2) by 2 ↑ n.

The error analysis in this computation is as follows:
Let ε be the machine precision then we have the following upper bounds in the successively computed values:

rel. error in $^2\log e$	:	ε ,
rel. error in $x \times {}^2\log e$	:	2ε ,
abs. error in y	:	$2\varepsilon \times \|x\| \times {}^2\log e$,
abs. error in 2 ↑ y	:	$2\varepsilon \times \|x\| + \varepsilon$,
rel. error in 2 ↑ y	:	$\varepsilon \times (2\|x\|+1) \times \sqrt{2}$,
rel. error in exp(x)	:	$\varepsilon \times (2\|x\|+1) \times \sqrt{2}$.

A few details in the procedure must be explained. They are due to the special features of the arithmetic of the EL X8.

a) exp(1447) is greater than the greatest real number which can be represented on the EL X8 (the giant) and exp(-1447) is smaller than the smallest positive representable number on the EL X8 (the dwarf). That is why the absolute value of the argument is bounded by 1447. Note: exp(1446) is smaller than the giant.

b) Before entering the procedure "two ttp" which evaluates the factor 2 ↑ n, we must check for exponent overflow; i.e. we must take care that this factor does not exceed 2 ↑ 2047. Because of the standardization of EL X8 real numbers the calculations for the case n > 2047 and for the case n < -2047 are not quite similar.

Remarks:

1. The relative error in exp(x) may be considerable for large values of $|x|$.

2. When $x \times {}^2\log e$ is integer the computation of the approximating polynomial is skipped. In particular this means that exp(0) = 1 holds exactly.

```
real procedure exp(x); value x; real x;
begin real two log e, two ttp 2047,
    c0, c1, c2, c3, c4, c5, c6, c7;
    integer e;
    boolean b;
    two log e    := +.1442695040889₁₀+1;
    two ttp 2047:= +.1615850303566₁₀+617;
    c0:= +.9999999999991;          c1:= +.6931471805237;
    c2:= +.2402265055080;          c3:= +.5550408537061₁₀-1;
    c4:= +.9617945040006₁₀-2;      c5:= +.1332563136000₁₀-2;
    c6:= +.1521326079998₁₀-3;      c7:= +.1283763199988₁₀-4;

    if abs(x) > 1447 then x:= sign(x) × 1447;
    x:= two log e × x;
    e:= entier(x) + 1;
    x:= x - e;
    if x = -1 then begin x:= 1; e:= e - 1; goto entire end;
    b:= x < -.5;
    if b then x:= x / 2;
    x:= ((((((c7 × x + c6) × x + c5) × x + c4) × x + c3) × x
    + c2) × x + c1) × x + c0;
    if b then x:= x × x;

entire: if e > 2047 then
    begin x:= x × two ttp 2047; e:= e - 2047 end;
    exp:= if e < -2047 then x / two ttp 2047 else x × two ttp(e)
end exp;
```

As an application of the exponential function we give the description in ALGOL 60 of the to-the-power function, as it is realized in the MILLI-system for the EL-X8.

```
real procedure ttp(x, y); value x, y; real x, y;
if y = 0 then ttp:= 1 else
if x = 0 ∧ y > 0 then ttp:= 0 else
if y ≠ entier(y) then ttp:= exp(ln(x) × y) else
if abs(y) < 32 then
begin real absy, t; absy:= abs(y); t:= 1;
loop: if entier(absy/2) × 2 ≠ absy then t:= t × x;
      absy:= entier(absy/2); if absy ≠ 0 then
      begin x:= x × x; goto loop end;
      ttp:= if y > 0 then t else 1/t
end else
ttp:= if x < 0 ∧ entier(y/2) × 2 ≠ y then
      -exp(ln(abs(x)) × y) else
       exp(ln(abs(x)) × y);
```

1.3. Natural logarithm

Two algorithms will be described. The first one has been taken from Barning [1965]. The second gives an alternative procedure which gives some better approximations in the neighbourhood of x = 1.

1.3.1 The algorithm from Barning [1965].

For the approximation of the natural logarithm we use the following elementary formulae:

$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \dots \quad , |x| < 1 ,$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots \quad , |x| < 1 ,$$

$$\ln\frac{1-x}{1+x} = -2x - \frac{2}{3}x^3 - \frac{2}{5}x^5 - \dots = \text{pol}(x) \quad . \qquad (1.3.1)$$

Let $y = \frac{1-x}{1+x}$. Then $x = \frac{1-y}{1+y}$, so that

$$\ln y = \text{pol}(\frac{1-y}{1+y}) , \quad y > 0 . \qquad (1.3.2)$$

For arbitrary y > 0 we can find an integer n such that $y = z \times 2 \uparrow n$ and $.5 \le z < 1$.

We know that $\ln y = n \times \ln 2 + \ln z$. In order to compute ln z we write $u = z \times (\frac{9}{8})^m$, m integer and $\frac{8}{9} \le u < 1$.
As a consequence we have $\ln y = n \times \ln 2 + m \times \ln\frac{8}{9} + \ln u$.
Our last step in the transformation of the argument is:

$$w = u \times \sqrt{\frac{9}{8}} , \quad \sqrt{\frac{8}{9}} \le w < \sqrt{\frac{9}{8}} . \qquad (1.3.3)$$

As a result we have $\ln y = n \times \ln 2 + m \times \ln\frac{8}{9} + \ln w - \frac{1}{2}\ln\frac{9}{8}$,

(1.3.4)

with $\ln w = \mathrm{pol}(\frac{1-w}{1+w}) = \mathrm{pol}\left(\frac{\sqrt{\frac{8}{9}} - u}{\sqrt{\frac{8}{9}} + u}\right)$.

Hence we have to look for an approximation of pol(x) in the interval

$$\left(\frac{1-\sqrt{\frac{9}{8}}}{1+\sqrt{\frac{9}{8}}}\ ,\ \frac{1-\sqrt{\frac{8}{9}}}{1+\sqrt{\frac{8}{9}}}\right] = (-17+12\sqrt{2},\quad 17-12\sqrt{2}] \approx (-.03,\ +.03]\ .$$

For the approximation of pol(x) in this interval we need only a 3-term polynomial

$$c_1\, x + c_2\, x^3 + c_3\, x^5.$$

The values of these coefficients, obtained by truncating the Chebyshev series, are given in the procedure text. This polynomial approximates pol(x) with a relative error which is less than the machine precision ε. We have the following error bounds for the successively computed values:

rel. error in u : ε
abs. error in u : ε

rel. error in $\left|\frac{\sqrt{\frac{8}{9}} - u}{\sqrt{\frac{8}{9}} + u}\right|$: $\frac{2\varepsilon\sqrt{\frac{8}{9}}(2+\sqrt{\frac{8}{9}})}{|\frac{8}{9} - u^2|} + \varepsilon \approx \frac{6\varepsilon}{|\frac{8}{9} - u^2|} + \varepsilon$

abs. error in $\left|\frac{\sqrt{\frac{8}{9}} - u}{\sqrt{\frac{8}{9}} + u}\right|$: $\frac{6\varepsilon}{(\sqrt{\frac{8}{9}} + u)^2} + \varepsilon\,\frac{\sqrt{\frac{8}{9}} - u}{\sqrt{\frac{8}{9}} + u} \leq$

$$\leq \frac{6\varepsilon}{(\sqrt{\frac{8}{9}} + \frac{8}{9})^2} + \varepsilon\,\frac{\sqrt{\frac{8}{9}} - \frac{8}{9}}{\sqrt{\frac{8}{9}} + \frac{8}{9}} \approx 2\varepsilon$$

abs. error in pol : $2 \times (2\varepsilon) + \varepsilon = 5\varepsilon$
abs. error in ln y : $\varepsilon \times m \times \ln\frac{8}{9} + 5\varepsilon + \varepsilon + \varepsilon \times n \times \ln 2$
$\approx \varepsilon \times (6 + 0.1m + 0.7n)$.

From this it is clear that ln x for $x \approx 1$ may have a great relative error and in fact one can observe that the computed value of $\ln(1-\varepsilon)$ is already wrong in the first digit, although the absolute error is small. One can see this immediately by observing that the value of ln 1 is obtained by subtracting $\frac{1}{2} \ln \frac{9}{8}$ from pol(x). For a good relative precision in a neighbourhood of zero the almost zero-value should be obtained by multiplication.

Remark:

In order to obtain ln(1) = 0 the value of the constant ln 8 over 9 had to be changed one bit in the least significant position.

1.3.2. An alternative algorithm

To overcome the difficulties mentioned above, we constructed a new algorithm.
First we transform the argument range $[1,\infty)$ into $[1,2)$ and $(0,1)$ into $[.5,1)$. We can always find an integer n such that y × 2 ↑ (-n) lies in one of the mentioned intervals (y denotes the argument). So, exactly multiplying by powers of 2, we have reduced the argument range $(0,\infty)$ to $[.5,2)$.
Let $y > 1$ and let k be an integer such that

$$2 \uparrow (1/2 \uparrow (k+1)) > y \geq 2 \uparrow (1/2 \uparrow k) .$$

Then we find:

$$2 \uparrow (1/2 \uparrow k) > y \times 2 \uparrow (-1/2 \uparrow k) \geq 1 \qquad (1.3.5)$$

and an analogous result holds for $y < 1$. Thus successively multiplying by numbers of the type 2 ↑ (±1/2 ↑ i), where $i \in \{1,2,\ldots,n\}$, we obtain a new argument range:

$$[2 \uparrow (-1/2 \uparrow n),\ 2 \uparrow (1/2 \uparrow n)) \quad .$$

This transformation is carried out with $n = 3$, thus reducing the interval to $[2^{-1/8}, 2^{1/8})$. In this interval a telescoped series of 4 terms is used, derived from (1.3.2).

$$\ln(x) = \ln(z) + (n \times 8 + j) \times \ln(2^{1/8}) ,$$

where n and j are integer, and

$$z \in [1,2^{1/8}), \quad n \geq 0, \quad 0 \leq j \leq 7, \quad \text{for } x \geq 1$$

and

$$z \in [2^{-1/8},1), \quad n \leq 0, \quad -7 \leq j \leq 0, \quad \text{for } 0 < x < 1 .$$

Error analysis.

We distinguish between the following four cases:

a) $1 \leq x < 2^{1/8}$.

In this case no tranformation on x will be involved and the error is due exclusively to the calculation of $\frac{x-1}{x+1}$ and of the polynomial. It is clear that x-1 will be calculated exactly, and that the error in x+1 will be bounded by $\varepsilon(x+1)$, with ε the machine precision. Hence the relative error in x+1 is bounded by ε, so the relative error in $(\frac{x-1}{x+1})$ is bounded by 2ε. The polynomial has been chosen such that the relative error in the result is bounded by ε and we find a relative error in $\ln(x)$ bounded by 4ε.

b) $2^{1/8} \leq x < 2$.

Let i denote the number of transformations that (1.3.5) used. The transformed argument y has a relative error bounded by $i \times \varepsilon$. Thus we do not calculate $\ln(y)$, but $\ln(y^*)$, where $y^* = y \pm$ abs. error(y); $\ln(y)$ being almost proportional to y-1 on $[1,2^{1/8})$, we calculate $\ln(y) \pm$ abs. error(y), where abs. error(y) = $y \times$ rel. error(y) $\approx \ln(y) \times$ rel.error(y) $\approx \ln^*(y) \times$ rel. error(y) $\leq i\varepsilon \ln^*(y)$. Hence the total relative error in $\ln^*(y)$ will be bounded by (see a) $(4+i)\varepsilon$.

We obtain:

$$\ln^*(x) = \ln^*(y) + \frac{k}{8}\ln 2 = \ln(y) \pm (4+i)\varepsilon \ln(y) + \frac{k}{8}\ln 2$$

$$= \ln(x) \pm (4+i)\varepsilon \{\ln(x) - \frac{k}{8}\ln 2\} \pm \varepsilon \ln(x).$$

Consequently the relative error is less than $(5+i)\varepsilon$.
In fact this bound may be reduced somewhat because $\{\ln(x) - \frac{k}{8}\ln 2\}$ is smaller than $\ln(x)/(k+1)$. Taking this into account we find:

$$\text{rel. error}(\ln(x)) \le \varepsilon + \frac{(4+i)}{k+1}\varepsilon \le 3.5\varepsilon.$$

c) $x \ge 2$.
We find in a similar way:

$$\ln^*(x) = \ln(x) \pm (4+i)\varepsilon \{\ln(x) - (\frac{k}{8} + n)\ln 2\} \pm \varepsilon \ln(x) .$$

Hence, $\text{rel. error}(\ln(x)) \le \varepsilon + \frac{(4+i)}{k+8n+1}\varepsilon < 2\varepsilon.$

d) For $x < 1$ we find similar results.

Resuming, we find that the relative error is bounded by 4ε, in particular we have $\ln(1) = 0$.

```
real procedure ln(x); value x; real x;
if x < 0 then ln:= -.1776646197514₁₀+629 else
begin integer n, m, sgn;
    real y, y1, f, f2, pol, ln2, ln8over9, halfln9over8,
    factor, c1, c2, c3, one;
    ln2          := .6931471805601;
    ln8over9     :=-.1177830356564;
    c1           := 2.000000000022;
    c2           := .6666664789391;
    c3           := .4004332758886;
    halfln9over8:= .5889151782816₁₀-1;
    one          := .9428090415822;

    n:= bin exp(x, sgn); m:= 0; y:= x; factor:= 1;
loop: if y < .8888888888887 then
    begin y:= y × 1.125; factor:= factor × 1.125;
        m:= m + 1; goto loop
    end;
    x:= x × factor;
    f:= (one - x) / (one + x); f2:= f × f;
    pol:= ((f2 × c3 + c2) × f2 + c1) × f;
    ln:= ln8over9 × m - pol - halfln9over8 + ln2 × n
end ln;

real procedure ln1(x); value x; real x;
if x < 0 then ln1:= -.1776646197514₁₀+629 else
begin real be, a, b, c, inva, invb, invc, x2, c0, c1, c2, c3, ln2;
    a   := 1.414213562373; inva:= .7071067811867;
    b   := 1.189207115003; invb:= .8408964152532;
    c   := 1.090507732666; invc:= .9170040432036;
    c0  := 2;               c1:= .6666666670335;
    c2  := .3999990212251;  c3:= .2865491631528;
    ln2:= .6931471805601;
    if x < 1 then
    begin be:= bin exp(x, x2); if x < inva then
        begin x:= x × a; be:= be - .5 end;
        if x < invb then
        begin x:= x × b; be:= be - .25 end;
        if x < invc then
        begin x:= x × c; be:= be - .125 end
    end
    else
    begin be:= bin exp(x, x2) - 1; x:= x × 2;
        if x > a then
        begin x:= x × inva; be:= be + .5 end;
        if x > b then
        begin x:= x × invb; be:= be + .25 end;
        if x > c then
        begin x:= x × invc; be:= be + .125 end
    end;
    x:= (x - 1) / (x + 1); x2:= x × x;
    ln1:= (((c3 × x2 + c2) × x2 + c1) × x2 + c0) × x + be × ln2
end ln1;
```

1.4. Sine and cosine

The computation of function values for the sine and cosine functions is performed by one and the same routine. The functions sine and cosine are defined for arguments x in the range $(-\infty,+\infty)$. In order to compute sin(x) or cos(x) this range is reduced to the interval $(-\pi/4, \pi/4)$.
By a proper choice of an integer k, depending on x, a number y can be obtained such that

$$x = \frac{\pi}{2}(y + k) , \quad y \in [-\tfrac{1}{2},+\tfrac{1}{2}] . \tag{1.4.1}$$

Now the next relationships perform the transformation from the infinite range into the finite interval.

$$\begin{aligned} \cos(x) = &\cos(\pi y/2) && \text{if } k \equiv 0 \pmod 4, \\ &\sin(\pi y/2) && \text{if } k \equiv 1 \pmod 4, \\ &-\cos(\pi y/2) && \text{if } k \equiv 2 \pmod 4, \\ &-\sin(\pi y/2) && \text{if } k \equiv 3 \pmod 4). \end{aligned} \tag{1.4.2}$$

$$\begin{aligned} \sin(x) = &\sin(\pi y/2) && \text{if } k \equiv 0 \pmod 4, \\ &-\cos(\pi y/2) && \text{if } k \equiv 1 \pmod 4, \\ &-\sin(\pi y/2) && \text{if } k \equiv 2 \pmod 4, \\ &\cos(\pi y/2) && \text{if } k \equiv 3 \pmod 4 . \end{aligned} \tag{1.4.3}$$

Moreover, since the sine is an odd function and cosine an even function, we have

$$\begin{aligned} \sin(\pi y/2) &= \text{sign}(y) \quad \sin(\text{abs}(\pi y/2)), \\ \cos(\pi y/2) &= \qquad\qquad\quad \cos(\text{abs}(\pi y/2)). \end{aligned} \tag{1.4.4}$$

Actually this reduces the range of computation of the sine or cosine to $[0,\pi/4)$.

In order to compute $\sin(\pi y/2)$ or $\cos(\pi y/2)$ for $y \in [0,\tfrac{1}{2}]$ polynomial approximations are used.

These polynomial approximations are chosen such that:

1. the odd (even) character of the sine (cosine) function is preserved ,

2. $\lim_{x \to 0} \frac{\sin(x)}{x} = 1$, $\left|\frac{\sin(x)}{x}\right| \leq 1$,

3. $\cos(0) = 1$, $|\cos(x)| \leq 1$,

4. optimal (relative) accuracy is attained, and

5. the number of multiplications used is minimal.

In order to fulfil requirement 1 for the sine function, the Taylorseries of $\sin(\pi y/2)$ is first rearranged such that a powerseries in $z = (\pi y/2)^2$ is multiplied by $\pi y/2$

$$\sin(\pi y/2) = \frac{\pi y}{2} \quad (1 - z/3! + z^2/5! - \dots) . \qquad (1.4.5)$$

With a view to requirement 2, the powerseries in z

$$\frac{\sin(\pi y/2)}{\pi y/2} - 1 = -z/3! + z^2/5! - \dots \qquad (1.4.6)$$

is truncated and the number of terms necessary for the required accuracy is decreased by economizing with Chebyshev polynomials.
We note that a high absolute precision of $\frac{\sin(x)}{x} - 1$ guarantees a high relative precision of $\sin(x)$.
The even character of the cosine function also leads us to write the Taylorseries of $\cos(\pi y/2)$ as a series in $z = (\pi y/2)^2$. Because of requirement 5, we apply the process of economizing to the powerseries

$$\cos(\pi y/2) - 1 = -z/2! + z^2/4! - z^3/6! + \dots . \qquad (1.4.7)$$

Aiming at a relative accuracy of 2^{-40}, both for the sine and the cosine, we need a truncated Taylorseries consisting of 8 terms. By economizing these series the number of terms can be reduced by two. The resulting

6-term polynomial approximation is used to calculate the function on the reduced interval.

As described in section 0.3 the computed coefficients of the polynomial approximations have been modified somewhat in order to correct for the rounding of the polynomial coefficients, which affects the final truncation error.

```
real procedure sincos(x, sin);
value x, sin; real x; bool sin;
begin real k, x2, f, two over pi,
    c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11;
    two over pi:= .63661 97723 673;
    c0 := +1;                       c1 := +.15707 96326 794₁₀+1;
    c2 := -.12337 00550 125₁₀+1; c3 := -.64596 40974 927₁₀+0;
    c4 := +.25366 95072 540₁₀+0; c5 := +.79692 62611 834₁₀-1;
    c6 := -.20863 46891 369₁₀-1; c7 := -.46817 52592 887₁₀-2;
    c8 := +.91916 54179 155₁₀-3; c9 := +.16042 92697 341₁₀-3;
    c10:= -.24856 34468 030₁₀-4; c11:= -.35564 00770 321₁₀-5;

    x:= x × two over pi;
    k:= entier(x + .5);
    x:= x - k;
    if sin then else k:= k + 1;
    k:= k - entier(k/4) × 4; comment k:= k(mod 4);
    x2:= x × x;
    f:= if entier(k/2) × 2 = k then
    (((((c11×x2+c9)×x2+c7)×x2+c5)×x2+c3)×x2+c1)×x else
     (((((c10×x2+c8)×x2+c6)×x2+c4)×x2+c2)×x2+c0;
    sincos:= if k > 1 then -f else f
end sincos;

real procedure sin(x); sin:= sincos(x, true);

real procedure cos(x); cos:= sincos(x, false);
```

1.5. Arcsine and arccosine

Of course the arcsine and arccosine functions can be computed by means of the arctangent function as follows

$$\arcsin(x) = \arctan \frac{x}{\sqrt{(1-x)(1+x)}} ,$$

$$\arccos(x) = \arctan \frac{\sqrt{(1-x)(1+x)}}{x} .$$

However, we will give here also an explicit algorithm.
In order to describe the computation of the arcsine and arccosine functions, it is sufficient to restrict ourselves to the computation of the arcsine function with an argument range [0,1]. Since arcsine is an odd function, the argument range can be extended to [-1,+1] in a trivial way. The computation of an arccosine value is easily reduced to the computation of an arcsine value by means of the transformations

$$\arccos(x) = \begin{cases} \frac{\pi}{2} - \arcsin(x) & \text{if } 0 < x \le \frac{1}{2}\sqrt{2} , \\ \arcsin \sqrt{1-x^2} & \text{if } \frac{1}{2}\sqrt{2} < x \le 1 . \end{cases} \qquad (1.5.1)$$

In fact the interval [0,1] is too large for finding a reasonable fast convergent series expansion for the arcsine function. However, by special transformations the argument range of a polynomial approximation can be reduced to a shorter interval. We will show here a transformation which yields an arbitrary short interval and prove that this transformation does not cause excessive increase of error.

The following transformation will be used:

$$\arcsin(u) = y + \arcsin(u \cos y - \sqrt{1-u^2} \sin y) ,$$

$$0 \le \sin y \le u , \qquad (1.5.2)$$

where y is an arbitrary fixed number, $0 \le y < \pi/2$, of which the cosine and sine values are known. With an appropriate choice of y we are able to make the expression

$$v = u \cos y - \sqrt{1-u^2} \sin y \qquad (1.5.3)$$

as small as we want. However, as a consequence of the conditions on y we find that $v \in [0,u]$. If a list of values of y with corresponding cosine and sine values is available, we can choose y from this list, such that sin y is the largest sine value not exceeding u.
The arcsine is approximated on the interval $[0,\sin(\pi/32)]$ using a 5-term polynomial derived from the Taylorseries of

$$\left(\frac{\arcsin(x)}{x} - \frac{\pi}{2}\right) \cdot \sqrt{1-x^2} \ .$$

We will now prove that the error induced by transformation (1.5.2) is not excessive. The error induced by a multiplication will be ε times the result of the multiplication. Thus we can give a bound for the error induced by the calculation of v:

$$\begin{aligned} \text{error}(v) &\le \varepsilon|u \cos y| + \varepsilon|\sqrt{1-u^2} \sin y| \\ &= \varepsilon(u \cos y + \sqrt{1-u^2} \sin y), \end{aligned} \qquad (1.5.4)$$

$$u,\ \sin y,\ \cos y \ge 0.$$

The relative error will be

$$\text{rel. error}(v) \le \varepsilon \frac{(u \cos y + \sqrt{1-u^2} \sin y)}{(u \cos y - \sqrt{1-u^2} \sin y)}$$

This is a monotonic increasing function for $y \in [0,u]$.
Since we only have to consider the case where $\sin y \approx u$, we can give as a bound for the error:

$$\text{error}(v) \le 2\varepsilon\, u \cos y,$$

where $v = u \cos y - \sqrt{1-u^2} \sin y$ is a small value.
The arcsine for this small argument is almost proportional to the argument.

In fact we calculate

$$\arcsin^*(u) = y + \arcsin(v + \mathrm{error}(v)) \approx y + v + \mathrm{error}(v)$$

instead of

$$\arcsin(u) = y + \arcsin(v) \approx y + v \ .$$

Consequently, $\mathrm{error}(\arcsin(u)) \approx \mathrm{error}(v)$.
Under the conditions $\sin y \approx u$ (i.e. $v \approx \arcsin(y)$) and $\sqrt{1-u^2}\,\sin y \leq y$, we find that

$$y + v = u \cos y - \sqrt{1-u^2}\,\sin y + y \geq u \cos y \ .$$

Now

$$\text{rel. error}(\arcsin^*(u)) \approx \frac{\mathrm{error}(v)}{\arcsin(u)} \leq \frac{2\varepsilon\, u \cos y}{v + y} \leq \frac{2\varepsilon\, u \cos y}{u \cos y} = 2\varepsilon \ . \qquad (1.5.5)$$

The main part of the calculation of an arcsine or an arccosine is performed by the real procedure arcsincos with parameters: x, y, sign, shift. This procedure calculates

$$\mathrm{sign} \times \arcsin(x) + \mathrm{shift} \times \pi/2$$

and presupposes the input parameter y to be equal to $\sqrt{1-x^2}$. The procedure uses transformation (1.5.2) and a table of sine values for $\pi/32, 2\pi/32, \ldots, 15\pi/32$.
arcsincos transforms the argument to an argument in the interval $[0, \sin(\pi/32)]$; it calculates the arcsine and performs the back transformation.

The procedure arcsin calculates the arcsine for the argument x; it first calculates $y = \sqrt{1-x^2}$ and calls arcsincos with appropriate sign, the shift being zero.

The procedure arccos calculates the arccosine for the argument x; it first calculates $y = \sqrt{1-x^2}$ and calls arcsincos. If $y < x$, arcsincos is called with y as the first parameter, otherwise with x as the first parameter. The parameters sign and shift are chosen appropriately in order to effectuate transformation (1.5.1).

```
real procedure arcsincos(x, y, sgn, shift);
value x, y, sgn, shift;
real x, y; integer sgn, shift;
begin real z, c0, c1, c2, c3, c4, pi over 2, pi over 32;
    integer i, count;
    array b[1:15];
    b[01]:= +.980171403296010-1; b[02]:= +.195090322016110-0;
    b[03]:= +.290284677254510-0; b[04]:= +.382683432365010-0;
    b[05]:= +.471396736826110-0; b[06]:= +.555570233019410-0;
    b[07]:= +.634393284163710-0; b[08]:= +.707106781186710-0;
    b[09]:= +.773010453363010-0; b[10]:= +.831469612302510-0;
    b[11]:= +.881921264348410-0; b[12]:= +.923879532510910-0;
    b[13]:= +.956940335731810-0; b[14]:= +.980785280403310-0;
    b[15]:= +.995184726672110-0; c0   := -.570796326794210-0;
    c1   := +.452064830058610-0; c2   := +.630162116800610-1;
    c3   := +.219830806334810-1; c4   := +.107099463170910-1;
    pi over 2:= 1.570796326794; pi over 32:= .981747704246310-1;
    count:= 0; i:= 8;
next: if x > b[count + i] then count:= count + i;
    i:= i : 2; if i ≠ 0 then goto next;
    if count ≠ 0 then
    begin z:= x; i:= 16 - count;
        x:= z × b[i] - y × b[count];
        y:= y × b[i] + z × b[count]
    end;
    z:= x × x;
    z:= (((((c4 × z + c3) × z + c2) × z + c1) × z + c0) /
    y + pi over 2) × x;
    arcsincos:= (count × pi over 32 + z) × sgn + shift × pi over 2
end arcsincos;
```

```
real procedure arcsin(x); value x; real x;
begin real y;
    integer sgn;
    sgn:= sign(x); x:= x × sgn; y:= sqrt((1 - x) × (1 + x));
    arcsin:= arcsincos(x, y, sgn, 0)
end arcsin;

real procedure arccos(x); value x; real x;
begin real y;
    integer sgn;
    y:= sqrt((1 - x) × (1 + x)); if x < y then
    begin sgn:= sign(x); x:= x × sgn; sgn:= - sgn;
        arccos:= arcsincos(x, y, sgn, 1)
    end
    else arccos:= arcsincos(y, x, 1, 0)
end arccos;
```

1.6. Tangent

The argument range $(-\infty,\infty)$ of the tangent function can be reduced to $[-\frac{\pi}{2}, \frac{\pi}{2}]$ by computing

$$x - \pi \text{ entier}(\frac{x}{\pi} + \frac{1}{2}) \text{ .} \tag{1.6.1}$$

We set $z = |\frac{8}{\pi} y|$ and using $\tan(-y) = -\tan(y)$ we find:
$\tan(y) = \pm \tan(\frac{\pi}{8} z)$, $z \in [0,4]$.
The Taylorseries of $\tan(\frac{\pi}{8} z)$ has convergence radius 4 and therefore is slowly convergent for z near 4. Hence we still have to reduce the interval. We use

$$\tan(\frac{\pi}{8} z) = \frac{1}{\tan(4- \frac{\pi}{8} z)} \tag{1.6.2}$$

and

$$\tan(\frac{\pi}{8} z) = \frac{1-\tan(2- \frac{\pi}{8} z)}{1+\tan(2- \frac{\pi}{8} z)} \text{ .} \tag{1.6.3}$$

Equation (1.6.2) will be used if $z > 2$, thus reducing the interval to $[0,2]$; equation (1.6.3) will be used if $z > 1$, reducing the interval to $[0,1]$. Although the convergence of the Taylorseries for the tangent function is guaranteed, the convergence is still very slow, due to the poles at $z = \pm 4$.
However, the Taylorseries of

$$(z+4)\ (z-4)\ \tan(\frac{\pi}{8} z) \tag{1.6.4}$$

yields better convergence. Hence $\tan(\frac{\pi}{8} z)$ will not be computed directly, but first $(z+4)(z-4)\tan(\frac{\pi}{8} z)$ is computed and we divide the result by (z^2-16).
Aiming at a maximal relative error of 2^{-40} we first truncated the Taylorseries and then economized this series to a polynomial consisting of 5 terms.

Error analysis.

We distinguish between the following three cases:

a. $0 \le x \le \pi/8$.

Here no transformations are needed. Since we first calculate $z = \frac{8}{\pi} x$, a relative error bounded by ε (ε denotes the machineprecision) is introduced.

Next we calculate $u = z^2$, $p(u) = \sum_{i=1}^{4} c_i u^i - 2\pi$, and we multiply $p(u)$ by z.

Since $\sum_{i=1}^{4} c_i u^i \ll 2\pi$, we find for $p(u)$ a relative error bounded by ε, and consequently a relative error bounded by 3ε for $p(u) \times z$.

We still have to compute $v = 16 - u$ and $\tan(x) = p(u) \times z/v$.

Since $u \le 1$ the relative error for v is bounded by ε.

So the relative error of $\tan(x)$ is bounded by 5ε.

b. $\pi/8 < x \le \pi/4$.

Now we use equation (1.6.3). Computing $z = 2 - \frac{8}{\pi} x$ we find an absolute error bounded by 2ε in z (the relative error may be large but this does not affect the result).

In fact we calculate $\tan^*(z+z')$ with $|z'| \le 2\varepsilon$.

The tangent function is about proportional to z for $z \in [0,\pi/8]$. Hence we find $\tan^*(z+z') \approx \tan^*(z) + z' \approx \tan(z) \times (1+t') + z'$, where $|t'| \le 5\varepsilon$ (cf. a).

The back transformation reads

$$\tan^*(x) = \frac{1 - \tan^*(z)}{1 + \tan^*(z)} \quad .$$

In the calculation of $1 - \tan^*(z)$, as with the calculation of $1 + \tan^*(z)$, errors are introduced.

Since $\tan^*(z) \le \tan(\pi/8) < .5$ we find

$$\text{rel error}(\tan^*(x)) \le \text{rel error}(1-\tan^*(z)) + \text{rel error}(1+\tan^*(z)) + \varepsilon$$

$$\le 4\varepsilon + 2\varepsilon + \varepsilon = 7\varepsilon \; .$$

c. $\pi/4 < x \leq \pi/2$.

In this case no rigorous bounds for the error can be given.
Calculating $z = 4 - \frac{8}{\pi} x$ we find an absolute error z' bounded by 4ε for z.

The back transformation reads

$$\tan^*(x) = \frac{1}{\tan^*(z)}, \quad \text{with } \tan^*(z) = \tan(z) \times (1+t') + z', \quad \text{with } |z'| \leq 4\varepsilon \text{ and } |t'| \leq 7\varepsilon.$$

We obtain

$$\text{rel error}(\tan^*(x)) \leq 7\varepsilon + \frac{4\varepsilon}{\tan(z)} .$$

However tan(z) may be very small.

The same arguments hold for $-\pi/2 \leq x < 0$.

```
real procedure tan(x); value x; real x;
begin integer sgn;
    real t, y, z, c0, c1, c2, c3, c4, one over pi;
    boolean b1, b2;
    c0:= -.6283185307177₁₀+1; c1:= +.6971703293846₁₀-1;
    c2:= +.2632215430363₁₀-3; c3:= +.1606181878952₁₀-5;
    c4:= +.1091903247058₁₀-7;
    one over pi:= .3183098861837;
    x:= x × one over pi; y:= entier(x + .5);
    x:= (x - y) × 8; sgn:= sign(x); x:= x × sgn;
    b1:= b2:= false;
    if x > 2 then
    begin x:= 4 - x; b1:= true end;
    if x > 1 then
    begin x:= 2 - x; b2:= true end;
    y:= x × x; z:= y - 16;
    t:= ((((c4 × y + c3) × y + c2) × y + c1) × y + c0) / z × x;
    if b2 then t:= (1 - t) / (1 + t);
    tan:= (if b1 then 1 / t else t) × sgn
end tan;
```

1.7. Arctangent

The argument range $(-\infty,+\infty)$ of the arctangent function can be reduced in the following way.

a) For arguments less than zero we use the relation $\arctan x = -\arctan(-x)$.

b) If $x > 1$ we compute an approximation to $\arctan(1/x)$ and use the relation

$$\arctan x = \frac{\pi}{2} - \arctan(1/x). \tag{1.7.1}$$

c) For $x \in [0,1]$, $\arctan x$ can be approximated by truncation of the Taylor-series

$$\arctan x = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} x^{2k+1} . \tag{1.7.2}$$

Because of the slow convergence of the series it is still necessary to reduce the argument range. This is possible by using the relation

$$\arctan x = \arctan y + \arctan \frac{x-y}{1+xy} . \tag{1.7.3}$$

Aiming at a precision of 2^{-40} it is sufficient to use the transformation (1.7.3) for only one value of y.
The best reduction is obtained for $y = \tan\frac{\pi}{6} = \frac{1}{3}\sqrt{3}$ and we find

$$\arctan x = \frac{\pi}{6} + \arctan \frac{x - \frac{1}{3}\sqrt{3}}{1 + \frac{x}{3}\sqrt{3}} . \tag{1.7.4}$$

We use this transformation for $x \in (\tan\frac{\pi}{12}, 1]$ and obtain the argument range $(-\tan\frac{\pi}{12}, \tan\frac{\pi}{12}] = (\sqrt{3}-2,\ 2-\sqrt{3}]$.
Putting $x^2 = z$ we can write the following power series of arctan x

$$\arctan x = x\ (1+z \sum_{k=1}^{\infty} \frac{(-1)^k}{2k+1} z^{k-1}) , \qquad |x| \le \tan\frac{\pi}{12} . \tag{1.7.5}$$

Aiming at an absolute error bounded by 2^{-40} in the approximation of (1.7.5), the necessary number of terms of $\sum_{k=1}^{n} \frac{(-1)^k}{2k+1} z^{k-1}$ can be decreased to $n = 6$ by economizing with Chebyshev polynomials.

Error analysis.

For an estimation of an upper bound of rounding errors arising during the computation of the value of arctan(x), we can distinguish between two cases,

a) $\quad |x| \le \tan \frac{\pi}{12}$.

In this case the value of the function is computed immediately by means of the polynomial approximation.
The last step in this computation reads

$$\arctan x = (p(z) \times z + 1) \times x \ .$$

The error in $p(z) \times z$ is suppressed by the addition by 1; indeed $|p(z) \times z| < 0.25$. As a result we find in arctan x a relative error bounded by 2ε (ε being the machineprecision).

b) $\quad |x| > \tan \frac{\pi}{12}$.

As a consequence of the computation of $1/x$ and $\frac{x - \tan(\pi/6)}{1 + x \tan(\pi/6)}$ we shall have an absolute error in the transformed argument bounded by 4ε.
The resulting absolute error of the computation only causes a small relative one because of the last addition in
$((p(z) \times z) + 1) \times x + \frac{\pi}{6}$ or in $((p(z) \times z) + 1) \times x + \frac{\pi}{3}$.
The total error is bounded by 5ε.

Resuming, we find that arctan(x) gives a small relative and absolute error on the whole range $(-\infty,+\infty)$.

```
real procedure arctan(x); value x; real x;
begin integer s; boolean xgr0;
    real c1, c2, c3, c4 ,c5, c6,
    tg 15, tg 30, pi over 6, x2, f;
    tg 15     := +.26794 91924 300;
    tg 30     := +.57735 02691 900;
    pi over 6:= +.52359 87755 987;
    c1:= -.33333 33332 462;     c2:= +.19999 99804 771;
    c3:= -.14285 54966 219;     c4:= +.11104 47077 384;
    c5:= -.89521 60021 931₁₀-1; c6:= +.62220 17887 490₁₀-1;

    xgr0:= x > 0; if ¬ xgr0 then x:= -x;
    s:= 3; if x > 1 then x:= 1/x else s:= s - 2;
    if x > tg 15 = s = 3 then s:= s - 1;
    if x > tg 15 then x:= (x - tg 30)/(1 + tg 30 × x);
    x2:= x × x;
    f:= ((((((c6 × x2 + c5) × x2 + c4) × x2 + c3) × x2
    + c2) × x2 + c1) × x2 + 1) × x;
    if s > 1 then f:= -f;
    f:= s × pi over 6 + f;
    arctan:= if xgr0 then f else -f
end arctan;
```

2. Double-length floating-point arithmetic

2.1. Elementary double-length operations

In this section we publish with the kind permission of the author, Prof.Dr. T.J. Dekker, the basic double-length procedures for addition, subtraction, multiplication, division and the evaluation of the square root. For a detailed description, one is referred to the original publication [1971].

A double-length floating-point number is a pair (r,s) of single-length floating-point numbers satisfying

$$|s| \leq |r+s| \frac{2^{-40}}{1+2^{-40}} .$$

The value of the double-length number (r,s) is, by definition, equal to r+s. We call r the head and s the tail of (r,s).

The procedures add2, sub2, mul2 and div2 calculate resp. the double-length sum, difference, product and quotient of (x,xx) and (y,y), the result being (z,zz). The procedure mul12 calculates the exact product of x and y, the result being the nearly double-length number (z,zz).

```
procedure add 2(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real r, s;
    r:= x + y;
    s:= if abs(x) > abs(y) then x − r + y + yy + xx
        else y − r + x + xx + yy;
    z:= r + s; zz:= r − z + s
end add 2;

procedure sub 2(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real r, s;
    r:= x − y;
    s:= if abs(x) > abs(y) then x − r − y − yy + xx
        else − y − r + x + xx −yy;
    z:= r + s; zz:= r − z + s
end sub 2;

procedure mul 12(x, y, z, zz);
value x, y; real x, y, z, zz;
begin real hx, tx, hy, ty, p, q;
    p:= x × 1048577; hx:= x − p + p; tx:= x − hx;
    p:= y × 1048577; hy:= y − p + p; ty:= y − hy;
    p:= hx × hy; q:= hx × ty + tx × hy;
    z:= p + q; zz:= p − z + q + tx × ty
end mul 12;

procedure mul 2(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real c, cc;
    mul 12(x, y, c, cc); cc:= x × yy + xx × y + cc;
    z:= c + cc; zz:= c − z + cc
end mul 2;

procedure div 2(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin real c, cc, u, uu;
    c:= x / y; mul 12(c, y, u, uu);
    cc:= (x − u − uu + xx − c × yy) / y;
    z:= c + cc; zz:= c − z + cc
end div 2;

procedure sqrt 2(x, xx, y, yy);
value x, xx; real x, xx, y, yy;
begin real c, cc, u, uu;
    if x > 0 then
    begin c:= sqrt(x); mul 12(c, c, u, uu);
        cc:= (x − u − uu + xx) × 0.5 / c;
        y:= c + cc; yy:= c − y + cc
    end else y:= yy:= 0
end sqrt 2;
```

2.2. Normalization

With the double length numbers as defined by Dekker [1971], some problems arise in the comparison of two double length numbers.
We can define equality ($\overset{*}{=}$) of two such numbers, X and Y, in two ways.

a. $X \overset{*}{=} Y$ iff $fl(X - Y) = 0$

or

b. $X \overset{*}{=} Y$ iff X and Y have equal representations.

Whatever definition will be chosen, the following condition must hold:

$$X \overset{*}{=} Y \rightarrow fl(X + C) \overset{*}{=} fl(Y + C) \quad \text{for arbitrary } C. \qquad (2.2.1)$$

Now let

$$(x,xx) = (2^{2t} + 2^{t+1}, -2^{t}) \;,$$
$$(y,yy) = (2^{2t}, + 2^{t}) \;,$$
$$(z,zz) = (2^{2t}, + 2^{t} - 1) \;,$$

where t denotes the bit-length of the mantissa.
We find: $(x,xx) = (y,yy) \neq (z,zz)$ and all three are acceptable double length numbers according to Dekker.
If we use definition b, we find that two equal numbers will not appear to be equal on a computer.
Definition a also gives difficulties. When we apply the procedure "sub 2" of Dekker on the pair {(x,xx), (z,zz)} the result will be zero, and the pair {(y,yy), (z,zz)} yields the result +1.
And so definition a gives: $(x,xx) \overset{*}{=} (z,zz) \overset{*}{\neq} (y,yy)$, however we find also $(x,xx) \overset{*}{=} (y,yy)$, and so for the equality defined by a, transitivity will not hold. Furthermore condition (2.2.1) does not hold because $fl(fl((x,xx) - 2^{2t}) - fl((z,zz) - 2^{2t})) = 1$.

With each definition we now propose a solution of the difficulties

a. $X \overset{*}{=} Y$ iff $fl(X - Y) = 0$.
Here the difficulty stems from the fact that the mantissa of the number (z,zz) contains more information than can be stored in 2t bits.

Thus we need a transformation, which transforms a double length number to a 2t-bit precision number.
If one of the following three conditions holds:

1. x and xx integral ; $2^{2t-1} < x < 2^{2t}$ and $|xx| < 2^t$,
2. x and xx integral ; $x = 2^{2t-1}$ and $0 \le xx < 2^t$,
3. x and 2.xx integral ; $x = 2^{2t-1}$ and $-2^{t-1} < xx < 0$,

then the pair (x,xx) is a 2t bit precision number.
Now consider a single length number y, $0 \le y < 2^t$; then on a computer with optimal addition and subtraction the number $z = fl(fl(y - 2^t) + 2^t)$ will be the number y rounded to the nearest integral value. For, if y is integral, the numbers $fl(y - 2^t)$ and z will be calculated exactly. Otherwise y must be smaller than 2^{t-1} and therefore $|fl(y - 2^t)| \ge 2^{t-1}$; so this result is integral and the same holds for z.
Now we will split up in the three cases mentioned above:

1. Let (x,xx) be a double length number with $2^{2t-1} < x < 2^{2t}$.
   Then in order to obtain a new tail of the double length number we have to carry out the calculation $fl(fl(xx - 2^t) + 2^t)$ for $xx \ge 0$ and $fl(fl(xx + 2^t) - 2^t)$ for $xx < 0$.
   If we calculate $v = x/2^t$ and $u = fl(x - v)$, then, on a computer with optimal subtraction, u will not be equal to x. Now x - u may not be smaller than 2^t, since both x and u are t-bit precision numbers. Furthermore let $x - u > 2^t$; then $x\text{-}fl(x - u) > 2^t$ and so $v > 2^t$ which is not true. Hence we find $x - u = 2^t$. If we calculate $y = fl(x - u)$ we find $y = x - u = 2^t \ne v$.
2. Let (x,xx) be a double length number with $x = 2^{2t-1}$ and $xx \ge 0$.
   If we calculate $v = x/2^t$ and $u = fl(x - v)$, both calculations will be exact and so $y = fl(x - u) = 2^{t-1} = v$.
   Here we have to multiply y by 2 before calculating the new tail $fl(fl(xx - y) + y)$.
3. Let (x,xx) be a double length number with $x = 2^{2t-1}$ and $xx < 0$.
   Again we find $y = 2^{t-1}$, and now we can calculate a new tail $fl(fl(xx-y)+y)$ directly, since 2.xx has to be an integral number.

Summarising we find the following algorithm:
calculate $v = x/2^t$, $u = fl(x - v)$ and $y = fl(x - u)$;
if $xx < 0$ set $y = -y$, otherwise if $y = v$ set $y = 2 * y$; calculate the new value of xx : $fl(fl(xx - y) + y)$.
Thus we round (x,xx) to a 2t-bit precision number. Because all calculations are homogeneous in x and xx, the same rounding procedure may be used for all double length numbers, only the test $xx < 0$ has to be changed into $x * xx < 0$.

b. $X \stackrel{*}{=} Y$ iff X and Y have equal representations.

Here the difficulty stems from the fact that double length numbers (x,xx), with xx a power of two that just cannot be incorporated in x, have two representations.
We have to decide in favor of one of these representations, and we choose the one where $|x|$ is maximal.
Now we have to be able to transform a number from the wrong representation into the favored representation.
The procedure we give only works on a computer with optimal rounding for the addition, where - if the result is just between two representable numbers - it is rounded away from zero if the operands have equal signs and towards zero otherwise.
We observe that if (x,xx) is a double length number with only one representation, then $fl(x + xx) = x$. If (x,xx) is a number with two representations then $|fl(x + xx)| < |x|$ if (x,xx) is the favored representation and $|fl(x + xx)| > |x|$ otherwise.
Hence we find the following algorithm:
calculate $y = fl(x + xx)$;
if $|y| > |x|$ set $xx = -xx$ and $x = y$.

2.3. Relational operations

Two sets of six Boolean procedures have been written in order to accomplish the comparison of double length numbers analogous to the six Boolean operators $<$, $\leq$, $>$, $\geq$, $=$ and $\neq$.
We again distinguish between the two definitions of equality given in section 2.2.

a. $X = Y$ iff $fl(X - Y) = 0$.

Here the following relation is used

$$\text{sign } fl((x,xx) - (y,yy)) = \text{sign}(fl(fl(fl(x - y) - yy) + xx)) ,$$

which holds for 2t-bit precision numbers.

b. $X \overset{*}{=} Y$ iff X and Y have equal representations.

Here the following relations are used:

$$(x,xx) \overset{*}{=} (y,yy) \equiv (x = y \wedge xx = yy),$$
$$(x,xx) \overset{*}{\neq} (y,yy) \equiv (x \neq y \vee xx \neq yy),$$
$$(x,xx) \overset{*}{<} (y,yy) \equiv (\text{if } x = y \text{ then } xx < yy \text{ else } x < y),$$
$$(x,xx) \overset{*}{\leq} (y,yy) \equiv (\text{if } x = y \text{ then } xx \leq yy \text{ else } x < y),$$
$$(x,xx) \overset{*}{>} (y,yy) \equiv (\text{if } x = y \text{ then } xx > yy \text{ else } x > y),$$
$$(x,xx) \overset{*}{\geq} (y,yy) \equiv (\text{if } x = y \text{ then } xx \geq yy \text{ else } x > y).$$

As procedure identifiers we have chosen: lng lt, lng le, lng gt, lng ge, lng eq and lng ne for the relations $<$, $\leq$, $>$, $\geq$, $=$ and $\neq$ respectively.

Procedures for normalization and relational operators as described under a.

```
procedure norm(x, xx); real x, xx;
if abs(x) = .17766461975514₁₀+629 ∨
   abs(x) < .68045389189205₁₀-604 then xx:= 0 else
begin real u, v;
    v:= x/1099511627776; u:= x - v; u:= x - u;
    if x < 0 = xx > 0 then u:= -u else
    if u = v then u:= u × 2;
    xx:= xx - u + u
end norm;
```

```
boolean procedure lng eq(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng eq:= x - y - yy + xx = 0;

boolean procedure lng ne(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng ne:= x - y - yy + xx ≠ 0;

boolean procedure lng gt(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng gt:= x - y - yy + xx > 0;

boolean procedure lng ge(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng ge:= x - y - yy + xx ≥ 0;

boolean procedure lng lt(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng lt:= x - y - yy + xx < 0;

boolean procedure lng le(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng le:= x - y - yy + xx ≤ 0;
```

Procedures for normalization and relational operators as described under b.

```
procedure norm(x, xx); value x, xx; real x, xx;
begin real y;
    y:= x + xx; if abs(y) > abs(x) then
    begin x:= y; xx:= -xx end
end norm;
```

```
boolean procedure lng eq(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng eq:= x = y ∧ xx = yy;

boolean procedure lng ne(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng ne:= x ≠ y ∨ xx ≠ yy;

boolean procedure lng gt(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng gt:= if x = y then xx > yy else x > y;

boolean procedure lng ge(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng ge:= if x = y then xx ≥ yy else x > y;

boolean procedure lng lt(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng lt:= if x = y then xx < yy else x < y;

boolean procedure lng le(x, xx, y, yy);
value x, xx, y, yy; real x, xx, y, yy;
lng le:= if  x = y then xx ≤ yy else x < y;
```

Procedures for double length addition, subtraction, multiplication or division.

For normalization either one of the procedures described under a or b may be used.

```
procedure lng add(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin add2(x, xx, y, yy, z, zz); norm(z, zz)
end lng add;

procedure lng sub(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin sub2(x, xx, y, yy, z, zz); norm(z, zz)
end lng sub;

procedure lng mul(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin mul2(x, xx, y, yy, z, zz); norm(z, zz)
end lng mul;

procedure lng div(x, xx, y, yy, z, zz);
value x, xx, y, yy; real x, xx, y, yy, z, zz;
begin div2(x, xx, y, yy, z, zz); norm(z, zz)
end lng div;
```

3. Input and output procedures

Because of the close resemblance to the single-length I/O procedures, the reader is assumed to be familiar with the I/O procedures of the MILLI system for the EL X8 computer as described in Grune [1972].

3.1. Input procedures

1. lng read 1 (intexpr,intvar,x,xx)

   The input procedure lng read 1 scans a row of symbols, of which the first is found in 'intvar' and the other ones by successive evaluations of 'intexpr'.
   When a correct number according to the Revised Report [1964] has been found, the procedure stores the last symbol, which does not belong to the number, in 'intvar' and transforms the row of symbols to a double precision number (x,xx), using triple length arithmetic; otherwise an error message is given. A tabulation, new line carriage return, two successive spaces or any other symbol, not being a point, a lower ten or digit, will be regarded as a number separator.

2. lng const (number,x,xx)

   The procedure lng const can be used when we need a constant in double precision in an ALGOL 60 program text.
   By a call of lng const (number,x,xx), the string 'number' is considered to be a real number and the double precision representation is delivered in the pair (x,xx).
   The number has to be written according to the specifications in the Revised Report with the following exceptions:
   a. two or more embedded blanks or
   b. an embedded new line carriage return
   will be regarded as the end of the number.
   The procedure makes use of the procedures printtext, stringsymbol [Grune,1972] and lng read 1.

3.2. Output procedures

In order to make the output procedures totally compatible with the input procedure lng read 1, all procedures in this section use triple length arithmetic.

User procedures:

A number of procedures is available to convert double length numbers to symbol strings.
The procedures are the analogs of flo, fix, fixt, flot, absfixt and print, respectively (cf. Grune [1972]). Only the most important features are recorded here.

1. lng flo (n,m,x,xx,a):

   The Boolean procedure lng flo stores the symbols of the double length number given in (x,xx) in floating format according to the specifications in n and m in the integer array a. The number will be rounded to n decimal places. If one of the following conditions holds:
   $n \leq 0$, $n > 25$, $m \leq 0$, $m > 3$ or the exponent is too big for m decimal positions, lng flo stores the number, according to n=25 and m=3, and delivers the value <u>false</u>; otherwise it delivers the value <u>true</u>.

2. lng fix (n,m,x,xx,a):

   The Boolean procedure lng fix stores the symbols of the double length number given in (x,xx) in fixed format according to the specifications in n and m in the integer array a. If one of the following conditions holds:
   $n < 0$, $m < 0$, $n + m = 0$, $n + m > 33$ or $|(x,xx)| \geq 10^n$,
   lng fix stores the number in floating format, according to n=25 and m=3, and delivers the value <u>false</u>; otherwise it delivers the value <u>true</u>.

3. lng fixt (n,m,x,xx).

4. lng flot (n,m,x,xx).

5. lng absfixt (n,m,x,xx).

6. lng print (x,xx):

   If the number is an integral number in absolute value smaller than 2^{80} the number is printed in fixed format, with 25 digits before and no digits after the decimal point, followed by 6 spaces.
   Otherwise the mantissa will be printed in floating format with 25 digits for the number and 3 digits for the exponent. In all cases the number takes 33 places on the printer.

Auxiliary procedures:

7. conbindec (x,xx,xxx,exp,sign):

   conbindec transforms the double length number given in $x^* = (x,xx)$ to a triple length number $x^{**} = (x,xx,xxx)$ with the following features:

   a. $.1 \leq x^{**} < 1$,

   b. $x^{**} \times \text{sign} \times 10^{\text{exp}} = x^*$.

8. round (n,z,zz,zzz,c,cc,ccc,dexp,dexp1)

   round adds $.5 \times 10^{-n}$ to the triple length number $z^* = (z,zz,zzz)$ yielding the triple length number $c^* = (c,cc,ccc)$.
   If $c^* \geq 1$, c^* is set to .1 and dexp1 will become the decimal exponent of z as given in dexp, increased by 1, otherwise dexp will be copied in dexp1.

9. nextchar (c,cc,ccc):

   If for the triple length number $c^* = (c,cc,ccc)$ the inequality $.1 \leq c < 1$ holds, the integer procedure nextchar delivers the first decimal digit of c^* and replaces c^* by $10 \times c^* - \text{entier}(10 \times c^*)$.

10. storeflo (c,cc,ccc,n,m,dexp,sgn,a):

    storeflo stores the symbols of the number $\text{sgn} \times c^* \times 10^{\text{dexp}}$, with $c^* = (c,cc,ccc)$, and $.1 \leq c^* < 1$ in floating format in the integer array a.

11. storefix (c,cc,ccc,n,m,dexp,sgn,a):

    storefix stores the symbols of the number $\text{sgn} \times c^* \times 10^{\text{dexp}}$, with $c^* = (c,cc,ccc)$, and $.1 \leq c^* < 1$, in fixed format in integer array a.

```
comment The procedure "lng bin exp" delivers the binary
        exponent of (x, xx) as an integer value.
        Moreover, the sign of (x, xx) is delivered in sgn
        and, if (x, xx) ≠ 0, (x, xx) is replaced by its binary
        mantissa (0.5 ≤ (x, xx) < 1).
        Although an efficient procedure is only possible
        in machine-code, an equivalent ALGOL-version is
        given below;
integer procedure lng bin exp(x, xx, sgn); real x; integer sgn;
begin integer i, e;
    sgn:= sign(x); if x < 0 then begin x:= -x; xx:= -xx end;
    if x = 0 then e:= 0 else
    if x < 1 then
    begin i:= e:= 0;
        for i:= i - 1 while x < 0.5 do
        begin e:= i; x:= x × 2; xx:= xx × 2 end
    end else
    for i:= 1, i + 1 while x ≥ 1 do
    begin e:= i; x:= x / 2; xx:= xx / 2 end;
    if if x = 0.5 then xx < 0 else false then
    begin x:= x × 2; xx:= xx × 2; e:= e - 1 end;
    lng bin exp:= e
end lng bin exp;


procedure conbindec(x, xx, xxx, exp, sgn); integer exp, sgn;
real x, xx, xxx;
begin integer bexp, dexp, s, ptr;
    real e, z, zz, zzz, c, cc, ccc, u, uu, v, vv;
    bexp:= lng binexp(x, xx, sgn); z:= x; zz:= xx; zzz:= 0;
    e:= .69388939039O7; s:= - 57; dexp:= 0;
    if bexp > 0 then
    begin
    n1: c:= z / e; mul12(e, c, u, uu);
        sub2(z, 0, u, uu, u, uu);
        add2(u, uu, zz, zzz, zz, zzz); cc:= zz / e;
        mul12(e, cc, u, uu); sub2(zz, zzz, u, uu, zzz, u);
        ccc:= zzz / e; z:= c + cc; zz:= c - z + cc + ccc;
        zzz:= c - z + cc - zz + ccc; bexp:= bexp + s;
        dexp:= dexp + 17;
        if if z > 1 then true else z = 1 ∧ zz > 0 then
        begin z:= z / 2; zz:= zz / 2; zzz:= zzz / 2;
            bexp:= bexp + 1
        end;
        if bexp > 0 then goto n1
    end;
    if bexp < 0 then
    begin ptr:= 17;
    n2: if bexp - s > 0 ∧ ptr > 1 then
        begin ptr:= ptr - 1; e:= e / 10;
            if ptr : 3 × 3 = ptr then
            begin e:= e × 16; s:= s + 4 end
            else
            begin e:= e × 8; s:= s + 3 end;
            goto n2
        end;
```

```
        mul12(e, z, u, uu); mul12(e, zz, v, vv);
        add2(u, uu, v, vv, c, cc); sub2(u, uu, c, 0, u, uu);
        add2(u, uu, v, vv, v, vv); u:= e × zzz;
        add2(v, vv, u, 0, cc, ccc); z:= c + cc;
        zz:= c - z + cc + ccc; zzz:= c - z + cc - zz + ccc;
        bexp:= bexp - s; dexp:= dexp - ptr;
        if if z < .5 then true else z = .5 ∧ zz < 0 then
        begin z:= z × 2; zz:= zz × 2; zzz:= zzz × 2;
            bexp:= bexp - 1
        end;
        if bexp < 0 then goto n2; if bexp ≠ 0 then
        begin e:= 2 ↑ bexp; z:= z × e; zz:= zz × e;
            zzz:= zzz × e;
            if if z > 1 then true else z = 1 ∧ zz > 0 then
            begin dexp:= dexp + 1; c:= z / 10;
                mul12(10, c, u, uu);
                sub2(z, 0, u, uu, u, uu);
                add2(u, uu, zz, zzz, zz, zzz); cc:= zz / 10;
                mul12(10, cc, u, uu);
                sub2(zz, zzz, u, uu, zzz, u); ccc:= zzz / 10;
                z:= c + cc; zz:= c - z + cc + ccc;
                zzz:= c - z + cc - zz + ccc
            end
        end
    end;
    x:= z; xx:= zz; xxx:= zzz; exp:= dexp
end conbindec;


procedure round(n, z, zz, zzz, c, cc, ccc, dexp, dexp1);
value n, z, zz, zzz, dexp; integer n, dexp, dexp1;
real z, zz, zzz, c, cc, ccc;
begin real u, uu;
    u:= 10 ↑ ( - n) / 2; add2(z, 0, u, 0, u, uu);
    add2(u, uu, zz, zzz, c, cc); sub2(u, uu, c, 0, u, uu);
    add2(u, uu, zz, zzz, cc, ccc);
    if if c > 1 then true else c = 1 ∧ cc > 0 then
    begin c:= .1; cc:= - .2273736754433₁₀ - 13;
        ccc:= .5169878828458₁₀ - 26; dexp1:= dexp + 1
    end
    else dexp1:= dexp
end round;


integer procedure nextchar(c, cc, ccc); real c, cc, ccc;
begin integer char;
    real u, uu, v, vv, z, zz, zzz;
    mul12(10, c, u, uu); mul12(10, cc, v, vv);
    add2(u, uu, v, vv, z, zz); sub2(u, uu, z, 0, u, uu);
    add2(u, uu, v, vv, v, vv); u:= ccc × 10;
    add2(v, vv, u, 0, zz, zzz); c:= z + zz;
    cc:= z - c + zz + zzz; ccc:= z - c + zz - cc + zzz;
    char:= entier(c);
    if char = c ∧ cc < 0 then char:= char - 1;
    nextchar:= char; sub2(c, 0, char, 0, u, uu);
    add2(u, uu, cc, ccc, c, zz); sub2(u, uu, c, 0, u, uu);
    add2(u, uu, cc, ccc, cc, ccc)
end nextchar;
```

```
procedure storeflo(c, cc, ccc, n, m, dexp, sgn, a);
value c, cc, ccc, n, m, dexp, sgn; integer n, m, dexp, sgn;
real c, cc, ccc; integer array a;
begin integer char, i, k;
    boolean zero;
    a[1]:= if sgn = 0 then 64 else 65; a[2]:= 88; i:= 3;
    for n:= n step - 1 until 1 do
    begin a[i]:= nextchar(c, cc, ccc); i:= i + 1 end;
    a[i]:= 89; if dexp > 0 then a[i + 1]:= 64 else
    begin a[i + 1]:= 65; dexp:= - dexp end;
    i:= i + 2; k:= 10 ↑ (m - 1); char:= dexp ÷ k; zero:= true;
    for m:= m - 1 step - 1 until 1 do
    begin if char ≠ 0 then zero:= false;
        a[i]:= if zero then 93 else char; i:= i + 1;
        dexp:= dexp - k × char; k:= k ÷ 10; char:= dexp ÷ k
    end;
    a[i]:= char; a[i + 1]:= 93
end storeflo;


procedure storefix(c, cc, ccc, n, m, dexp, sgn, a);
value c, cc, ccc, n, m, dexp, sgn; integer n, m, dexp, sgn;
real c, cc, ccc; integer array a;
begin integer i, j;
    j:= n - dexp; if j > n then j:= n; i:= 1; n:= n - j;
    for j:= j step - 1 until 1 do
    begin a[i]:= 93; i:= i + 1 end;
    if m = 0 ∧ n = 0 then
    begin a[i - 1]:= if sgn = 0 then 64 else 65; a[i]:= 0
    end
    else a[i]:= if sgn = 0 then 64 else 65;
    for n:= n step - 1 until 1 do
    begin i:= i + 1; a[i]:= nextchar(c, cc, ccc) end;
    if m ≠ 0 then
    begin i:= i + 1; a[i]:= 88 end;
    j:= - dexp; if j < 0 then j:= 0 else if j > m then j:= m;
    m:= m - j;
    for j:= j step - 1 until 1 do
    begin i:= i + 1; a[i]:= 0 end;
    for m:= m step - 1 until 1 do
    begin i:= i + 1; a[i]:= nextchar(c, cc, ccc) end;
    a[i + 1]:= 93
end storefix;


boolean procedure lng flo(n, m, x, xx, a); value n, m, x, xx;
integer n, m; real x, xx; integer array a;
begin integer exp, exp1, sgn;
    real xxx, c, cc, ccc;
    conbindec(x, xx, xxx, exp, sgn);
    if n < 0 ∨ n > 25 ∨ m < 0 ∨ m > 3 then
    begin lng flo:= false; n:= 25; m:= 3 end
```

```
    else lng flo:= true;
    round(n, x, xx, xxx, c, cc, ccc, exp, exp1);
    if if m = 1 then abs(exp1) > 9 else if m = 2 then
    abs(exp1) > 99 else false then
    begin lng flo:= false; n:= 25; m:= 3;
        round(n, x, xx, xxx, c, cc, ccc, exp, exp1)
    end;
    storeflo(c, cc, ccc, n, m, exp1, sgn, a)
end lng flo;


boolean procedure lng fix(n, m, x, xx, a); value n, m, x, xx;
integer n, m; real x, xx; integer array a;
begin integer exp, exp1, sgn;
    real xxx, c, cc, ccc;
    conbindec(x, xx, xxx, exp, sgn);
    if n < 0 ∨ m < 0 ∨ n + m = 0 ∨ n + m > 33 then
flo:
    begin lng fix:= false;
        round(25, x, xx, xxx, c, cc, ccc, exp, exp1);
        storeflo(c, cc, ccc, 25, 3, exp1, sgn, a)
    end
    else
    begin lng fix:= true;
        round(exp + m, x, xx, xxx, c, cc, ccc, exp, exp1);
        if exp1 > n then goto flo;
        storefix(c, cc, ccc, n, m, exp1, sgn, a)
    end
end lng fix;


procedure lng flot(n, m, x, xx); value n, m, x, xx;
integer n, m; real x, xx;
begin integer i, max;
    integer array a[1:33];
    max:= if lng flo(n, m, x, xx, a) then n + m + 5 else 33;
    if max + printpos > 144 then nlcr;
    for i:= 1 step 1 until max do prsym(a[i])
end lng flot;


procedure lng fixt(n, m, x, xx); value n, m, x, xx;
integer n, m; real x, xx;
begin integer i, max;
    integer array a[1:36];
    max:= if lng fix(n, m, x, xx, a) then (if m = 0 then n +
    2 else n + m + 3) else 33;
    if max + printpos > 144 then nlcr;
    for i:= 1 step 1 until max do prsym(a[i])
end lng fixt;
```

```
procedure lng absfixt(n, m, x, xx); value n, m, x, xx;
integer n, m; real x, xx;
begin integer i, max;
    integer array a[1:36];
    if lng fix(n, m, x, xx, a) then
    begin max:= if m = 0 then n + 2 else n + m + 3; i:= 0;
        for i:= i + 1 while a[i] = 93 do ; a[i]:= 93
    end
    else max:= 33; if max + printpos > 144 then nlcr;
    for i:= 1 step 1 until max do prsym(a[i])
end lng absfixt;


procedure lng print(x, xx); value x, xx; real x, xx;
begin real c, cc;
    if x < 0 then
    begin c:= - x; cc:= - xx end
    else
    begin c:= x; cc:= xx end;
    if printpos > 111 then nlcr;
    if if c > .120892581961510 + 25 then true else if c =
    .120892581961510 + 25 ∧ cc > 0 then true else c ≠
    entier(c) ∨ cc ≠ entier(cc) then lng flot(25, 3, x, xx)
    else
    begin lng fixt(25, 0, x, xx); space(6) end
end lng print;


procedure lng read1(intexpr, intvar, x, xx);
integer intexpr, intvar; real x, xx;
begin integer exp, exp1, sbl, sgn, sgne;
    real m, mm, mmm, u, uu, v, vv, z, zz, zzz, e;
    boolean point;

    integer procedure nextsym;
    begin sbl:= intexpr;
        if sbl = 93 then sbl:= intexpr else if sbl = 120
        then
        for sbl:= intexpr while sbl ≠ 120 do ; nextsym:= sbl
    end nextsym;

    sbl:= intvar; exp:= exp1:= 0;
start: sgn:= sgne:= 0; point:= false;
start1: if sbl = 64 ∨ sbl = 65 then
    begin sgn:= sbl - 64; nextsym end;
    if sbl = 93 then
    begin nextsym; goto start1 end;
    if sbl = 88 then
    begin point:= true;
        goto if nextsym < 10 ∧ sbl ≥ 0 then number else
        start
    end;
```

```
    if sbl = 89 then
a1: begin if nextsym = 93 then goto a1;
        if sbl = 64 ∨ sbl = 65 then
        begin sgne:= sbl - 64; nextsym end;
    b1: if sbl = 93 then
        begin nextsym; goto b1 end;
        goto if sbl < 10 ∧ sbl ≥ 0 then exponent else start
    end;
    if sbl ≥ 10 ∨ sbl < 0 then
    begin nextsym; goto start end;
number: m:= sbl; mm:= mmm:= 0; if point then exp1:= 1;
number1: if nextsym = 88 then
    begin if point then goto ready; point:= true;
        if nextsym ≥ 10 ∧ sbl < 0 then
        begin nlcr; nlcr; printtext(⪡er 517⪢); exit end
    end;
    if sbl < 10 ∧ sbl ≥ 0 then
    begin mul12(10, m, u, uu); mul12(10, mm, v, vv);
        add2(u, uu, v, vv, z, zz); sub2(u, uu, z, 0, u, uu);
        add2(u, uu, v, vv, v, vv); u:= mmm × 10;
        add2(v, vv, u, 0, zz, zzz);
        add2(zzz, 0, sbl, 0, u, uu);
        add2(u, uu, z, zz, m, mm); sub2(z, zz, m, 0, z, zz);
        add2(u, uu, z, zz, mm, mmm); z:= m + mm;
        zz:= m - z + mm + mmm; zzz:= m - z + mm - zz + mmm;
        m:= z; mm:= zz; mmm:= zzz;
        if point then exp1:= exp1 + 1; goto number1
    end;
    if sbl = 89 then
a2: begin if nextsym = 93 then goto a2;
        if sbl = 64 ∨ sbl = 65 then
        begin sgne:= sbl - 64; nextsym end;
    b2: if sbl = 93 then
        begin nextsym; goto b2 end;
        if sbl < 10 ∧ sbl ≥ 0 then goto exponent1; nlcr; nlcr;
        printtext(⪡er 517⪢); exit
    end;
    goto ready;
exponent: m:= 1; mm:= mmm:= 0;
exponent1: exp:= sbl;
exponent2: if nextsym < 10 ∧ sbl ≥ 0 then
    begin exp:= exp × 10 + sbl; goto exponent2 end;
ready: if sgne = 1 then exp:= - exp; exp:= exp - exp1;
    if sgn = 1 then
    begin m:= - m; mm:= - mm; mmm:= - mmm end;
    e:= ₁₀17; intvar:= sbl; sbl:= 17;
    if abs(exp) > 1000 then exp:= sign(exp) × 1000;
    if exp > 0 then
    begin
```

```
    n1: if exp < sbl then
        begin e:= e / 10; sbl:= sbl - 1; goto n1 end;
        mul12(e, m, u, uu); mul12(e, mm, v, vv);
        add2(u, uu, v, vv, z, zz); sub2(u, uu, z, 0, u, uu);
        add2(u, uu, v, vv, v, vv); u:= e × mmm;
        add2(v, vv, u, 0, zz, zzz); m:= z + zz;
        mm:= z - m + zz + zzz; mmm:= z - m + zz - mm + zzz;
        exp:= exp - sbl; if exp ≠ 0 then goto n1
    end
    else if exp < 0 then
    begin
    n2: if exp > - sbl then
        begin e:= e / 10; sbl:= sbl - 1; goto n2 end;
        z:= m / e; mul12(e, z, u, uu);
        sub2(m, 0, u, uu, u, uu);
        add2(u, uu, mm, mmm, mm, mmm); zz:= mm / e;
        mul12(e, zz, u, uu); sub2(mm, mmm, u, uu, mmm, u);
        zzz:= mmm / e; m:= z + zz; mm:= z - m + zz + zzz;
        mmm:= z - m + zz - mm + zzz; exp:= exp + sbl;
        if exp ≠ 0 then goto n2
    end;
    norm(m, mm); x:= m; xx:= mm
end lng read1;


procedure lng read(x, xx); real x, xx;
begin integer gts;
    gts:= 119; lng read1(resym, gts, x, xx)
end lng read;


procedure lng const(c, x, xx); real x, xx; string c;
begin integer gts, i;
    boolean last;

    integer procedure next; if last then
    begin nlcr; nlcr; printtext(⟨×er 517⟩) end
    else
    begin gts:= next:= stringsymbol(i, c); last:= gts = 255;
        i:= i + 1
    end next;

    last:= false; i:= 0; gts:= 119;
    lng read1(next, gts, x, xx)
end lng const;
```

3.3. Fast output procedure

In this section we describe an output procedure for double length real numbers which only uses elementary double length arithmetic operations as described in section 2.1 and single length output procedures which are available in the MILLI system for the EL X8.
Thus the procedure does not use explicitly the binary representation of the double length numbers nor some triple length arithmetic in order to obtain a printout that is correct in 25 digits.
As a consequence we only obtain a printout of 24 or fewer digits with the possibility of a little loss of accuracy in the 24-th digit. However, this output is about 3 times as fast as the output obtained by lng flot described in section 3.2. In addition,we may remark that there is no reason to assume that the result of any double length computation of which the output is wanted will be more accurate than the number printed by fast lng flot.

fast lng flot (n,m,x,xx):

This procedure prints the value of (x,xx) in floating point format. It is a double length analog of the single length procedure flot (n,m,x). In the case where only a single length printout is wanted (i.e. $1 \leq n \leq 12$, $1 \leq m \leq 3$), a call of fast lng flot results in a call of flot.
The procedure fast lng flot uses the procedures:
flot, fix, printpos, nlcr, prsym (available in the MILLI system) and sub 2 (see section 2.1) and the procedures lng entier and lng mul ttp 10 .

Auxiliary procedures:

The procedure lng entier (x,xx,y,yy) delivers in (y,yy) the largest integer value less than or equal to (x,xx).
The procedure lng mul ttp 10 (ep,m,mm,z,zz) delivers in (z,zz) the value of (m,mm) multiplied by 10^{ep}.

```
procedure fast lng flot(n, m, x, xx); value n, m, x, xx;
integer n, m; real x, xx;
if n < 1 ∨ n > 24 ∨ m < 1 ∨ m > 3 then
fast lng flot(24, 3, x, xx)
else if n < 13 then flot(n, m, x) else
begin integer i, sgn, sym;
    real e, ee, z, zz, mp, mmp, ep;
    integer array amp, ammp, aep[1:21];
    sgn:= 64; if x = 0 then
    begin for i:= 2 step 1 until 14 do amp[i]:= ammp[i]:= 0;
        fix(m, 0, 0, aep); goto pr
    end
    else
    begin if x < 0 then
        begin sgn:= 65; x:= - x; xx:= - xx end;
        ep:= entier(ln(x) × 0.4342944819033 + 0.99999);
    aa: lng mul ttp 10(12 - ep, x, xx, z, zz);
        lng entier(z, zz, mp, mmp); if mp ≥ ₁₀12 then
        begin ep:= ep + 1; goto aa end;
        if mp < ₁₀11 then mp:= ₁₀11 else sub2(z, zz, mp, 0,
        mmp, zz);
    end;
bb: if ¬fix(m, 0, ep, aep) then
    begin m:= 3; n:= 24; fix(m, 0, ep, aep) end;
cc: if ¬fix(0, n - 12, mmp, ammp) then
    begin mp:= mp + 1; mmp:= 0; goto cc end;
    if ¬fix(12, 0, mp, amp) then
    begin ep:= ep + 1; mp:= ₁₀11; goto bb end;
pr: if n + m + printpos > 139 then nlcr; i:= 1;
    for sym:= sgn, 88, amp[i - 1] while i < 14, ammp[i - 12]
    while i ≤ n + 2, 89, if ep > 0 then 64 else 65, if
    aep[i - n - 3] > 9 then 93 else aep[i - n - 3] while
    i ≤ n + m + 5 do
    begin prsym(sym); i:= i + 1 end
 end fast lng flot;


procedure lng entier(x, xx, y, yy); value x, xx;
real x, xx, y, yy;
begin real z, zz;
    z:= entier(x); if z = x then zz:= entier(xx) else zz:= 0;
    y:= z + zz; yy:= z - y + zz
end lng entier;
```

```
procedure lng mul ttp 10(ep, m, mm, z, zz); value ep, m, mm;
integer ep; real m, mm;
begin integer ab;
    real x, xx, y, yy;
    ab:= abs(ep); xx:= yy:= 0;
    if ab < 13 then y:= 10 ↑ ab else
    begin x:= 10; y:= if even(ab) + 1 = 0 then x else 1;
    loop: ab:= ab : 2; if ab ≠ 0 then
        begin mul2(x, xx, x, xx, x, xx);
            if even(ab) + 1 = 0 then mul2(y, yy, x, xx, y,
            yy); goto loop
        end
    end;
    if ep < 0 then div2(m, mm, y, yy, z, zz) else mul2(m,
    mm, y, yy, z, zz); norm(z, zz)
end lng mul ttp 10;
```

4. The computation of double length elementary functions

4.1. Long square root

The square root of a double precision number is calculated by use of the procedure sqrt2 [Dekker, 1971]. The result is normalized by one of the procedures norm (see chapter 2).

```
procedure lng sqrt(x, xx, y, yy);
value x, xx; real x, xx, y, yy;
begin sqrt2(x, xx, y, yy);
    norm(y, yy)
end lng sqrt;
```

4.2. Long exponential function

The method used for the computation of the double length exponential function is essentially the same as the one used for single length computation (see sect. 1.2).
The main difference between the two algorithms is the interval on which 2^x is approximated by a polynomial.

The algorithm for the long exponential function reads as follows:
First we reduce the argument range to [-1,0);
indeed we may write:

$$\exp(x) = 2 \uparrow (x \times {}^{2}\log e) = 2 \uparrow (n + y)$$

for $y \in [-1,0)$ and a certain integer n.
Next we reduce the interval [-1,0) to $[-2^{-k},0)$ for some integer k, which has to be chosen in advance.
In order to achieve this we have to divide the argument by 2^i for an integer $i \le k$.
On the interval $[-2^{-k},0)$ we calculate the value of 2^x by a polynomial approximation, derived by economizing the Taylorseries for 2^x

$$2^x = 1 + x \ln 2 + (x \ln 2)^2 / 2! + \dots .$$

The resulting value has to be squared i times successively.
For the single length exponential function, k = 1 has been chosen;
for the double length exponential function, k = 3 appears to be optimal.
For the approximation of 2^x on the interval [-0.125,0) we need a polynomial of degree 10 to obtain a relative truncation error bounded by 2^{-80}.

From the range of the argument and the magnitude of the coefficients, it is evident that some of the higher order terms only need to be calculated in single precision.

Since most values of the constants used are not clear from the ALGOL 60 procedure text we will give these coefficients here.

$^{2}\log e \approx 1.44269\ 50408\ 88963\ 40735\ 99247$

$c_0 = 1.00000\ 00000\ 00000\ 00000\ 00000$

$c_1 = 0.69314\ 71805\ 59945\ 30941\ 71869$

$c_2 = 0.24022\ 65069\ 59100\ 71231\ 88733$

$c_3 = 0.05550\ 41086\ 64821\ 57812\ 03680$

$c_4 = 0.00961\ 81291\ 07628\ 35979\ 04492$

$c_5 = 0.00133\ 33558\ 14638\ 45891\ 73248$

$c_6 = 0.00015\ 40353\ 03831\ 64485\ 52140$

$c_7 = 0.00001\ 52527\ 32274\ 79$

$c_8 = 0.00000\ 13215\ 33964\ 62$

$c_9 = 0.00000\ 01016\ 92817\ 83$

$c_{10} = 0.00000\ 00067\ 56094\ 68$

Note: Since the elementary double length procedures, as e.g. add2 and mul2, fail in case of overflow or underflow, the argument is tested at the beginning of the procedure and in situations where overflow (or underflow) may be expected, the giant (respectively, the dwarf) is delivered as value of the double length exponential function.

```
procedure lng exp(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
if x > .1446598165827₁₀+4 then
begin z:= .1776646197514₁₀+629; zz:= 0 end else
if x < -.1418179131426₁₀+4 then
begin z:= .6188692094765₁₀-616; zz:= 0 end else
begin real twologe, ttwologe, tp2047, t, tt,
           c6, cc6, c7, c8, c9, c10;
    integer e, i, m;
    array c, cc[0 : 5];
    tp2047:=  .1615850303566₁₀+617;
    twologe:= .1442695040889₁₀+1; ttwologe:= .1701065226463₁₀-12;
    c10 :=    .6756094679934₁₀-8;
    c 9 :=    .1016928178251₁₀-6;
    c 8 :=    .1321533964619₁₀-5;
    c 7 :=    .1525273227479₁₀-4;
    c 6 :=    .1540353038316₁₀-3;     cc 6 :=+.2151715844927₁₀-16;
    c[5]:=    .1333355814639₁₀-2;     cc[5]:=-.4334941696900₁₀-15;
    c[4]:=    .9618129107622₁₀-2;     cc[4]:=+.6306592747748₁₀-14;
    c[3]:=    .5550410866482₁₀-1;     cc[3]:=+.1327269032995₁₀-14;
    c[2]:=    .2402265069591;         cc[2]:=+.3948667549177₁₀-13;
    c[1]:=    .6931471805601;         cc[1]:=-.1723944453014₁₀-12;
    c[0]:=     1;                     cc[0]:=  0;

    mul2(twologe, ttwologe, x, xx, x, xx);
    e:= entier(x) + 1; sub2(x, xx, e, 0, x, xx);
    if x > 0 then begin e:= e + 1; sub2(x, xx, 1, 0, x, xx) end;
    if if x = -1 then xx = 0 else false then
    begin x:= 1; e:= e - 1; goto entire end;
    m:= 0;
again: if x < -.125 then
    begin x:= x / 2; xx:= xx / 2; m:= m + 1; goto again end;
    add2((((c10 × x + c9) × x + c8) × x + c7) × x, 0,
            c6, cc6, t, tt);
    for i:= 5 step -1 until 0 do
    begin mul2(t, tt, x, xx, t, tt);
        add2(t, tt, c[i], cc[i], t, tt)
    end;
    x:= t; xx:= tt;
    for m:= m step -1 until 1 do mul2(x, xx, x, xx, x, xx);
entire: if e > 2047 then
    begin mul2(x, xx, tp2047, 0, x, xx); e:= e - 2047 end;
    if e < -2047 then div2(x, xx, tp2047, 0, z, zz) else
    mul2(x, xx, two ttp(e), 0, z, zz);
    norm(z, zz)
end lng exp;
```

4.3. Long natural logarithm

The method used for the calculation of the double precision logarithm is essentially the same as the one described in section 1.3.2.

The argument range $(0,\infty)$ is reduced to $[0.5,1)$ for arguments smaller than 1 and to $[1,2)$ for arguments greater than 1.
These two argument ranges are reduced further by multiplication (if appropriate) by $\sqrt{2}$, $\sqrt[4]{2}$ and $\sqrt[8]{2}$ for arguments smaller than 1 and by $1 / \sqrt{2}$, $1 / \sqrt[4]{2}$ and $1 / \sqrt[8]{2}$ for arguments greater than 1.
This results in an argument range $[1 / \sqrt[8]{2}, \sqrt[8]{2})$.
On this range the logarithm is calculated and a multiple of ln(2) is added to compensate for the multiplication by powers of two in the reduction of the argument range.
The logarithm is calculated using an economized polynomial for the Taylor-series

$$\ln(x) = y(2 + \frac{2}{3}y^2 + \frac{2}{5}y^4 + \ldots) ,$$

where

$$y = \frac{x-1}{x+1} , \quad y \in \left[-\frac{\sqrt[8]{2}-1}{\sqrt[8]{2}+1}, \frac{\sqrt[8]{2}-1}{\sqrt[8]{2}+1} \right] \approx [-0.04, 0.04] .$$

In order to obtain a relative precision of 2^{-80} an economized polynomial of 7 terms is needed.
Because of the range of the argument and the magnitude of the coefficients, it is not necessary to compute the 2 higher order terms in double precision. In the procedure the following constants are used

cf_0 = 2.00000 00000 00000 00000 00000
cf_1 = 0.66666 66666 66666 66659 70638
cf_2 = 0.40000 00000 00000 59401 99634
cf_3 = 0.28571 42857 12384 71552 94079
cf_4 = 0.22222 22251 18845 55496 99236
cf_5 = 0.18181 59166 571
cf_6 = 0.15472 39939 787

$$\ln 2 = 0.69314\ 71805\ 59945\ 30941\ 72321$$

$$a = 1.41421\ 35623\ 73095\ 04880\ 16887 \approx \sqrt{2}$$

$$b = 1.18920\ 71150\ 02721\ 06671\ 74997 \approx \sqrt[4]{2}$$

$$c = 1.09050\ 77326\ 65257\ 65920\ 70108 \approx \sqrt[8]{2}$$

$$inva = 0.70710\ 67811\ 86547\ 52440\ 08443 \approx 1\ /\ \sqrt{2}$$

$$invb = 0.84089\ 64152\ 53714\ 54303\ 11259 \approx 1\ /\ \sqrt[4]{2}$$

$$invc = 0.91700\ 40432\ 04671\ 23174\ 35420 \approx 1\ /\ \sqrt[8]{2}$$

cf_0 to cf_6 are the coefficients of the polynomial.

```
procedure lng ln(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
if x < 0 then
begin z:= -.1776646197514⏨+629; zz:= 0 end else
begin integer i;
    real be, a, aa, b, bb, c, cc, inva, iinva, invb,
         iinvb, invc, iinvc, x2, xx2, y, yy,
         ln2, lln2, cf4, ccf4, cf5, cf6;
    array cf, ccf[0:3];
    a     := .1414213562373⏨+1; aa := -.2387946232276⏨-12;
    b     := .1189207115003⏨+1; bb := -.5068880128918⏨-12;
    c     := .1090507732666⏨+1; cc := -.8253702643269⏨-12;
    inva := .7071067811867;  iinva:= -.1193973116138⏨-12;
    invb := .8408964152541;  iinvb:= -.3502343692188⏨-12;
    invc := .9170040432045;  iinvc:= +.1286912640906⏨-12;
    cf[0]:=  2;              ccf[0]:=  0;
    cf[1]:= .6666666666670; ccf[1]:= -.3031649701938⏨-12;
    cf[2]:= .4000000000001; ccf[2]:= -.9035545021391⏨-13;
    cf[3]:= .2857142857124; ccf[3]:= -.1704544549108⏨-13;
    cf 4 := .2222222251189; ccf 4 := -.6676491564324⏨-13;
    cf 5 := .1818159166571;
    cf 6 := .1547239939787;
    ln2  := .6931471805601; lln2  := -.1723944452559⏨-12;

    if if x < 1 then true else
       if x > 1 then false else xx < 0 then
    begin be:= lng binexp(x, xx, x2);
        if x - inva - iinva + xx < 0 then
        begin mul2(x, xx, a, aa, x, xx); be:= be - .5 end;
        if x - invb - iinvb + xx < 0 then
        begin mul2(x, xx, b, bb, x, xx); be:= be - .25 end;
        if x - invc - iinvc + xx < 0 then
        begin mul2(x, xx, c, cc, x, xx); be:= be - .125 end
    end
    else
    begin be:= lng binexp(x, xx, x2) - 1; x:= x × 2; xx:= xx × 2;
        if x - a - aa + xx ≥ 0 then
        begin mul2(x, xx, inva, iinva, x, xx); be:= be + .5 end;
        if x - b - bb + xx ≥ 0 then
        begin mul2(x, xx, invb, iinvb, x, xx); be:= be + .25 end;
        if x - c - cc + xx ≥ 0 then
        begin mul2(x, xx, invc, iinvc, x, xx); be:= be + .125 end
    end;
    sub2(x, xx, 1, 0, y, yy); add2(x, xx, 1, 0, x, xx);
    div2(y, yy, x, xx, x, xx); mul2(x, xx, x, xx, x2, xx2);
    add2((cf6 × x2 + cf5) × x2, 0, cf4, ccf4, y, yy);
    for i:= 3 step - 1 until 0 do
    begin mul2(y, yy, x2, xx2, y, yy);
        add2(cf[i], ccf[i], y, yy, y, yy)
    end;
    mul2(y, yy, x, xx, y, yy); mul2(ln2, lln2, be, 0, x, xx);
    add2(x, xx, y, yy, z, zz); norm(z, zz)
end lng ln;
```

4.4. Long sine and long cosine

The method used for the computation of the double length sine and cosine functions is the same as used for the single length computation. Again, the argument range is reduced to $[0, \pi/4)$ and, again, polynomial approximations to the sine and the cosine functions are used on this interval.

In order to obtain the required relative accuracy of 2^{-80} for sin(x), 10 terms are required for the approximation of the series

$$\frac{\sin(\pi y/2)}{\pi y/2} - 1 = -z/3! + z^2/5! - \ldots, \quad 0 \le z < \pi^2/16$$

(equation 1.4.6).

By economizing the 10-term polynomial, the number of terms can be reduced by 2. Because of the range of the argument and the magnitude of the polynomial coefficients it is not necessary to compute (in the Horner scheme) the 2 higher order terms with double precision.

In order to obtain the required accuracy for cos(x), 11 terms are necessary for the approximation of

$$\cos(\pi y/2) - 1 = -z/2! + z^2/4! - \ldots, \quad 0 \le z < \pi^2/16$$

(equation 1.4.7).

Economizing again reduces the number of terms by 2. Because of the magnitude of the coefficients, here it is not necessary to compute (in the Horner scheme) the 3 higher order terms with double precision.

Since the coefficients of the approximating polynomials are not available in a decimal notation from the ALGOL 60 text, we will give these coefficients here.
The coefficients are identified analogously to the notation used in the procedure sincos (see sect. 1.4)

c_0 = +1.00000 00000 00000 00000 000
c_2 = −1.23370 05501 36169 82735 431
c_4 = +0.25366 95079 01048 01363 650
c_6 = −0.02086 34807 63352 96086 618
c_8 = +0.00091 92602 74839 42629 795
c_{10} = −0.00002 52020 42373 05479 743
c_{12} = +0.00000 04710 87477 81468 613
c_{14} = −0.00000 00063 86602 62795 386
c_{16} = +0.00000 00000 65657 82677 443
c_{18} = −0.00000 00000 00525 58615 920

c_1 = +1.57079 63267 94896 61923 132
c_3 = −0.64596 40975 06246 25365 494
c_5 = +0.07969 26262 46167 04503 305
c_7 = −0.00468 17541 35318 68450 865
c_9 = +0.00016 04411 84787 28591 491
c_{11} = −0.00000 35988 43234 35782 842
c_{13} = +0.00000 00569 21723 41883 667
c_{15} = −0.00000 00006 68780 54774 899
c_{17} = +0.00000 00000 06017 88412 179

```
procedure lng sincos (x, xx, z, zz, sin);
value x, xx; real x, xx, z, zz; boolean sin;
begin real n, nn, x2, xx2, y, yy, two over pi, two over pii,
           c 0, cc 0, c 1, cc 1, c 14, c 15, c 16, c 17, c 18;
    integer i;
    array c[2:13], cc[2:13];
    two over pi := +.6366197723673;
    two over pii:= +.241878239712210-12;
    c  0 :=   1;                     cc  0 :=   0;
    c  1 := +.157079632679410+ 1; cc  1 := +.744354748046410-12;
    c[ 2]:= -.123370055013610+ 1; cc[ 2]:= -.333589312790610-12;
    c[ 3]:= -.645964097506310- 0; cc[ 3]:= +.476569006051510-13;
    c[ 4]:= +.253669507901110- 0; cc[ 4]:= -.744015689652110-13;
    c[ 5]:= +.796926262461310- 1; cc[ 5]:= +.386609509366110-13;
    c[ 6]:= -.208634807633510- 1; cc[ 6]:= +.208732660998310-16;
    c[ 7]:= -.468175413531910- 2; cc[ 7]:= +.553721645385710-15;
    c[ 8]:= +.919260274839610- 3; cc[ 8]:= -.146546076685210-15;
    c[ 9]:= +.160441184787310- 3; cc[ 9]:= -.218185606632610-16;
    c[10]:= -.252020423730510- 4; cc[10]:= -.333876311209910-17;
    c[11]:= -.359884323435810- 5; cc[11]:= +.732448474234810-21;
    c[12]:= +.471087477814610- 6; cc[12]:= +.455993793892010-19;
    c[13]:= +.569217234188510- 7; cc[13]:= -.130383427560710-19;
    c 14 := -.638660262795510- 8;
    c 15 := -.668780547748810- 9;
    c 16 := +.656578267744010-10;
    c 17 := +.601788412178910-11;
    c 18 := -.525586159192610-12;

    mul 2(x, xx, two over pi, two over pii, x,  xx);
    add 2(x, xx, .5, 0, n, nn); lng entier(n, nn, n, nn);
    sub 2(x, xx, n, nn, x, xx);
    if ¬ sin then add 2(n, nn, 1, 0, n, nn);
    lng entier(n / 4, nn / 4, x2, xx2);
    sub 2(n, nn, 4 × x2, 4 × xx2, n, nn);
    mul 2(x, xx, x, xx, x2, xx2);
    if even(n) = 1 then
    begin mul12(c 17 × x2 + c 15, x2, y, yy);
        for i:= 13 step -2 until 3 do
        begin add 2(y, yy, c[i], cc[i], y, yy);
            mul 2(y, yy, x2, xx2, y, yy)
        end;
        add 2(y, yy, c 1, cc 1, y, yy);
        mul 2(y, yy, x, xx, y, yy)
    end else
    begin mul12((c 18 × x2 + c 16) × x2 + c 14, x2, y, yy);
        for i:= 12 step -2 until 2 do
        begin add 2(y, yy, c[i], cc[i], y, yy);
            mul 2(y, yy, x2, xx2, y, yy)
        end;
        add 2(y, yy, c 0, cc 0, y, yy)
    end;
    if n > 1 then begin z:= -y; zz:= -yy end else
    begin z:= y; zz:= yy end;
end lng sincos;
```

```
procedure lng sin(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
begin lng sincos(x, xx, z, zz, true); norm(z, zz) end lng sin;

procedure lng cos(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
begin lng sincos(x, xx, z, zz, false); norm(z, zz) end lng cos;
```

4.5. Long arcsine and long arccosine

The procedures written to calculate the double precision arcsine and arccosine functions use the same method as the procedures described in section 1.5.

The argument range is reduced to $[0,\frac{\pi}{32}]$. On this interval the arcsine is calculated, using a truncated Taylorseries of

$$\frac{\sqrt{1-x^2}\ \mathrm{arcsine}(x)}{x} .$$

Aiming at a precision of 2^{-80} this polynomial can be economized to 9 terms. Because of the range of the argument and the magnitude of the coefficients, it is not necessary to compute the 4 higher order terms in double precision.

In the following list we give the decimal representation of the constants used in the procedure.

c_0	=	-.57079 63267 94896 61923 13211
c_1	=	.45206 48300 64114 97628 21997
c_2	=	.06301 62075 16028 74442 41618
c_3	=	.02198 42942 34204 47064 45192
c_4	=	.01056 55807 21976 78865 31589
c_5	=	.00601 06250 59318
c_6	=	.00379 75788 19974
c_7	=	.00257 67633 29682
c_8	=	.00190 42820 79931
b_1	=	.09801 71403 29560 60199 41961
b_2	=	.19509 03220 16128 26784 82860
b_3	=	.29028 46772 54462 36763 61922
b_4	=	.38268 34323 65089 77172 84601
b_5	=	.47139 67368 25997 64855 63877
b_6	=	.55557 02330 19602 22474 28310
b_7	=	.63439 32841 63645 49821 51712
b_8	=	.70710 67811 86547 52440 08443
b_9	=	.77301 04533 62736 96081 09073
b_{10}	=	.83146 96123 02545 23707 87886
b_{11}	=	.88192 12643 48355 02971 27569
b_{12}	=	.92387 95325 11286 75612 81824
b_{13}	=	.95694 03357 32208 86493 57976
b_{14}	=	.98078 52804 03230 44912 61820
b_{15}	=	.99518 47266 72196 88624 48369
pi over 2	=	1.57079 63267 94896 61923 13211
pi over 32	=	.09817 47704 24681 03870 19576

c_0 to c_8 are the coefficients of the approximating polynomial and b_i (i=1,...,15) are the values of $\sin(\frac{i\pi}{32})$.

```
procedure lng arcsincos(x, xx, y, yy, sgn, shift, z, zz);
value x, xx, y, yy, sgn, shift; real x, xx, y, yy, z, zz;
integer sgn, shift;
begin real x2, xx2, u, uu, v, vv, f, ff, pi over 2, ppi over 2,
    pi over 32, ppi over 32, c 4, cc 4, c 5, c 6, c 7, c 8;
    integer i, count;
    array b, bb[1:15], c[0:3], cc[0:3];
    b[01]:= +.9801714032960⏨-1; bb[01]:= -.4202357606385⏨-13;
    b[02]:= +.1950903220161⏨-0; bb[02]:= +.4292988439940⏨-13;
    b[03]:= +.2902846772545⏨-0; bb[03]:= -.1389671578668⏨-13;
    b[04]:= +.3826834323650⏨-0; bb[04]:= +.1333832456361⏨-12;
    b[05]:= +.4713967368261⏨-0; bb[05]:= -.1388268725512⏨-12;
    b[06]:= +.5555702330194⏨-0; bb[06]:= +.2496252300453⏨-12;
    b[07]:= +.6343932841637⏨-0; bb[07]:= -.6149593466274⏨-13;
    b[08]:= +.7071067811867⏨-0; bb[08]:= -.1193973116138⏨-12;
    b[09]:= +.7730104533630⏨-0; bb[09]:= -.2486115011199⏨-12;
    b[10]:= +.8314696123025⏨-0; bb[10]:= +.4651975211767⏨-13;
    b[11]:= +.8819212643484⏨-0; bb[11]:= -.6563402400368⏨-13;
    b[12]:= +.9238795325109⏨-0; bb[12]:= +.3772714288139⏨-12;
    b[13]:= +.9569403357318⏨-0; bb[13]:= +.3937256384016⏨-12;
    b[14]:= +.9807852804033⏨-0; bb[14]:= -.9412836554843⏨-13;
    b[15]:= +.9951847266721⏨-0; bb[15]:= +.1462849077319⏨-12;
    c[00]:= -.5707963267951⏨-0; cc[00]:= +.1651399537249⏨-12;
    c[01]:= +.4520648300640⏨-0; cc[01]:= +.6901247330551⏨-13;
    c[02]:= +.6301620751606⏨-1; cc[02]:= -.2822126315042⏨-13;
    c[03]:= +.2198429423422⏨-1; cc[03]:= -.1096951130264⏨-13;
    c  4 := +.1056558072197⏨-1; cc  4 := +.3455341833452⏨-14;
    c  5 := +.6010625059318⏨-2;
    c  6 := +.3797578819974⏨-2;
    c  7 := +.2576763329682⏨-2;
    c  8 := +.1904282079931⏨-2;
    pi over 2  := .1570796326794⏨+ 1;
    ppi over 2 := .7443547480480⏨-12;
    pi over 32 := .9817477042463⏨- 1;
    ppi over 32:= .4652217175300⏨-13;

    count:= 0; i:= 8;
next: if x > b[count + i] then count:= count + i;
    i:= i : 2; if i ≠ 0 then goto next;
    if count ≠ 0 then
    begin x2:= x; xx2:= xx; i:= 16 - count;
        mul2(x2, xx2, b[i], bb[i], u, uu);
        mul2(y, yy, b[count], bb[count], v, vv);
        sub2(u, uu, v, vv, x, xx);
        mul2(y, yy, b[i], bb[i], u, uu);
        mul2(x2, xx2, b[count], bb[count], v, vv);
        add2(u, uu, v, vv, y, yy)
    end;
    mul2(x, xx, x, xx, x2, xx2);
    add2((((c 8 × x2 + c 7) × x2 + c 6) × x2 + c 5) × x2, 0,
            c 4, cc 4, f, ff);
    for i:= 3 step -1 until 0 do
    begin mul2(f, ff, x2, xx2, f, ff);
        add2(f, ff, c[i], cc[i], f, ff)
    end;
```

```
    div2(f, ff, y, yy, f, ff);
    add2(f, ff, pi over 2, ppi over 2, f, ff);
    mul2(f, ff, x, xx, f, ff);
    if count ≠ 0 then
    begin mul2(count, 0, pi over 32, ppi over 32, u, uu);
        add2(f, ff, u, uu, f, ff)
    end;
    if sgn = - 1 then begin f:= - f; ff:= - ff end;
    if shift = 1 then add2(f, ff, pi over 2, ppi over 2, z, zz)
    else begin z:= f; zz:= ff end
end lng arcsincos;

procedure lng arcsin(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
begin real u, uu, y, yy;
    integer sgn;
    sgn:= sign(x); if sgn = - 1 then
    begin x:= - x; xx:= - xx end;
    sub2(1, 0, x, xx, u, uu); add2(1, 0, x, xx, y, yy);
    mul2(u, uu, y, yy, y, yy); sqrt2(y, yy, y, yy);
    lng arcsincos(x, xx, y, yy, sgn, 0, z, zz);
    norm(z, zz)
end lng arcsin;

procedure lng arccos(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
begin real u, uu, y, yy;
    integer sgn;
    sub2(1, 0, x, xx, u, uu); add2(1, 0, x, xx, y, yy);
    mul2(u, uu, y, yy, y, yy); sqrt2(y, yy, y, yy);
    if x < y then
    begin sgn:= - sign(x); if sgn =  1 then
        begin x:= - x; xx:= - xx end;
        lng arcsincos(x, xx, y, yy, sgn, 1, z, zz)
    end
    else lng arcsincos(y, yy, x, xx, 1, 0, z, zz);
    norm(z, zz)
end lng arccos;
```

4.6. Long tangent

For the double precision approximation of the tangent function we use the same method as used for the single precision computation (see sect. 1.6). That is, the argument range $(-\infty,\infty)$ is reduced to $[0,\frac{\pi}{8}]$ by means of the transformations (1.6.1), (1.6.2) and (1.6.3), respectively.

In order to obtain a relative precision of 2^{-80} in this interval we use 9 terms of the economized Taylorseries of $(z+4)(z-4)\tan(\frac{\pi}{8} z)$, of which the last 3 terms are used in single precision only.

In the following list we give the decimal representation of the constants used in the procedure.

two pi = 6.28318 53071 79586 47692 52842 $\approx 2\pi$
inv pi = 0.31830 98861 83790 67153 77674 $\approx \frac{1}{\pi}$
c_1 = 0.06971 70329 45601 02797 97352
c_2 = 0.00026 32214 85528 43415 24042
c_3 = 0.00000 16063 42902 33349 38770
c_4 = 0.00000 00107 35005 74954 93508
c_5 = 0.00000 00000 73610 68764 74039
c_6 = 0.00000 00000 00508 95927 98524
c_7 = 0.00000 00000 00003 52752 43509
c_8 = 0.00000 00000 00000 02527 29791

```
procedure lng tan(x, xx, y, yy);
value x, xx; real x, xx, y, yy;
begin integer i;
    boolean xneg, xgr1, xgr2;
    real twopi, twopii, invpi, invpii, z, zz, t, tt,
         x2, xx2, c6, c7, c8;
    array c, cc[1 : 5];

    twopi:= .62831 85307 177₁₀+ 1; twopii:= +.29774 18992 192₁₀-11;
    invpi:= .31830 98861 837₁₀- 0; invpii:= +.12093 91198 561₁₀-12;
    c[1] := .69717 03294 562₁₀- 1; cc[1] := -.17456 53875 558₁₀-13;
    c[2] := .26322 14855 285₁₀- 3; cc[2] := -.52301 68212 832₁₀-16;
    c[3] := .16063 42902 333₁₀- 5; cc[3] := +.74697 44662 161₁₀-18;
    c[4] := .10735 00574 955₁₀- 7; cc[4] := -.68027 13806 076₁₀-21;
    c[5] := .73610 68764 739₁₀-10; cc[5] := +.18392 69750 547₁₀-22;
    c6   := .50895 92798 524₁₀-12;
    c7   := .35275 24350 857₁₀-14;
    c8   := .25272 97912 297₁₀-16;

    mul 2(x, xx, invpi, invpii, x, xx);
    add 2(x, xx, 0.5, 0, t, tt);
    lng entier(t, tt, z, zz); sub 2(x, xx, z, zz, x, xx);
    x:= x × 8; xx:= xx × 8; xneg:= x<0;
    if xneg then begin x:= -x; xx:= -xx end;
    xgr2:= x>2 ∨ x=2 ∧ xx>0;
    if xgr2 then sub 2(4, 0, x, xx, x, xx);
    xgr1:= x>1 ∨ x=1 ∧ xx>0;
    if xgr1 then sub 2(2, 0, x, xx, x, xx);

    mul 2(x, xx, x, xx, x2, xx2); sub 2(x2, xx2, 16, 0, z, zz);
    mul12((c8 × x2 + c7) × x2 + c6, x2, t, tt);
    for i:= 5 step -1 until 1 do
    begin add 2(t, tt, c[i], cc[i], t, tt);
        mul 2(t, tt, x2, xx2, t, tt)
    end;
    sub 2(t, tt, twopi, twopii, t, tt);
    mul 2(t, tt, x, xx, t, tt);
    div 2(t, tt, z, zz, y, yy);

    if xgr1 then
    begin sub 2(1, 0, y, yy, z, zz);
        add 2(1, 0, y, yy, t, tt); div 2(z, zz, t, tt, y, yy)
    end;
    if xgr2 then div 2(1, 0, y, yy, y, yy);
    if xneg then begin y:= -y; yy:= -yy end;
    norm(y, yy)
end lng tan;
```

4.7. Long arctangent

When we compute a double precision approximation to the arctangent function, it is possible to map the argument range $(-\infty,\infty)$ onto $[0,1]$ in the same way as we did in the single precision case.
For a further reduction we use the relation

$$\arctan x = \arctan y_k + \arctan \frac{x-y_k}{1+xy_k} ,$$

for one of the values y_k

$$y_k = \tan \frac{k\pi}{2(2n-1)} , \quad k=0,1,\dots,n-1 .$$

We choose k such that

$$\tan \frac{(2k-1)\pi}{4(2n-1)} < x \le \tan \frac{(2k+1)\pi}{4(2n-1)} ,$$

which yields the reduced argument range

$$\left|\frac{x-y_k}{1+xy_k}\right| \le \tan \frac{\pi}{4(2n+1)} .$$

For the single precision computation this transformation was used with $n = 2$.

If n is large we have the advantage of a small argument range and we need to evaluate only a small number of terms in the truncated power series. However in this case a large number of constants $\tan \frac{(2k-1)\pi}{4(2n-1)}$ and $\tan \frac{k\pi}{2(2n-1)}$ have to be stored.

In order to obtain optimal efficiency we use the transformation with $n = 5$. In this case we need 10 terms of the economized Taylorseries, of which the last 3 terms are used in single precision.

In the following list we give the decimal representation of the constants used in the procedure.

t_1 = 0.17632 69807 08464 97347 10901 ≈ tan (π / 18)
t_2 = 0.36397 02342 66202 36135 10476 ≈ tan (2π / 18)
t_3 = 0.57735 02691 89625 76450 91487 ≈ tan (3π / 18)
t_4 = 0.83909 96311 77280 01176 31260 ≈ tan (4π / 18)
tg_5 = 0.08748 86635 259 ≈ tan (π / 36)
tg_{15} = 0.26794 91924 311 ≈ tan (3π / 36)
tg_{25} = 0.46630 76581 550 ≈ tan (5π / 36)
tg_{35} = 0.70020 75382 097 ≈ tan (7π / 36)
pi o 18 = 0.17453 29251 99432 95769 23690 ≈ π / 18
c_0 = 1.00000 00000 00000 00000 00000
c_1 = -0.33333 33333 33333 33333 33333
c_2 = 0.19999 99999 9999 9999 93288
c_3 = -0.14285 71428 57142 85480 34634
c_4 = 0.11111 11111 11107 97121 78020
c_5 = -0.09090 90909 06979 38113 91253
c_6 = 0.07692 30761 25159 72129 04831
c_7 = -0.06666 64894 3985
c_8 = 0.05880 05593 3909
c_9 = -0.05102 12017 5872

```
procedure lng arctan(x, xx, z, zz);
value x, xx; real x, xx, z, zz;
begin integer i, s;
    boolean xneg, xgr1;
    real tg 5, tg 15, tg 25, tg 35, pio18, piloo18,
         x2, xx2, aux, auxx, c7, c8, c9;
    array c, cc[1 : 6], t, tt[1 : 4];

    tg 5 := +.87488 66352 592₁₀-1; tg 15 := +.26794 91924 311;
    tg 25:= +.46630 76581 550;     tg 35 := +.70020 75382 097;
    t[1] := +.17632 69807 084;     tt[1] := +.83375 81992 138₁₀-13;
    t[2] := +.36397 02342 662;     tt[2] := +.40209 88026 725₁₀-13;
    t[3] := +.57735 02691 900;     tt[3] := -.36634 01478 456₁₀-12;
    t[4] := +.83909 96311 773;     tt[4] := -.25115 73091 116₁₀-13;
    pio18:= +.17453 29251 994;     pioo18:= +.32178 59968 459₁₀-13;
    c[1] := -.33333 33333 335;     cc[1] := +.15158 24502 956₁₀-12;
    c[2] := +.20000 00000 000;     cc[2] := -.45474 73575 986₁₀-13;
    c[3] := -.14285 71428 571;     cc[3] := -.32479 61424 100₁₀-13;
    c[4] := +.11111 11111 111;     cc[4] := +.22123 84840 680₁₀-13;
    c[5] := -.90909 09090 696₁₀-1; cc[5] := -.17516 46373 007₁₀-13;
    c[6] := +.76923 07612 513₁₀-1; cc[6] := +.33221 89885 650₁₀-13;
    c7   := -.66666 48943 985₁₀-1;
    c8   := +.58800 55933 909₁₀-1;
    c9   := -.51021 20175 872₁₀-1;

    xneg:= x<0; if xneg then begin x:= -x; xx:= -xx end;
    xgr1:= x>1 ∨ x=1 ∧ xx>0;
    if xgr1 then div 2(1, 0, x, xx, x, xx);
    aux:= x + xx;
    s:= if aux < tg 5  then 0 else if aux < tg 15 then 1 else
        if aux ≤ tg 25 then 2 else if aux ≤ tg 35 then 3 else 4;
    if s > 0 then
    begin sub 2(x, xx, t[s], tt[s], aux, auxx);
        mul 2(x, xx, t[s], tt[s], x, xx);
        add 2(x, xx, 1, 0, x, xx);
        div 2(aux, auxx, x, xx, x, xx)
    end;

    mul 2(x, xx, x, xx, x2, xx2);
    mul12((c9 × x2 + c8) × x2 + c7,x2, aux, auxx);
    for i:= 6 step -1 until 1 do
    begin add 2(aux, auxx, c[i], cc[i], aux, auxx);
        mul 2(aux, auxx, x2, xx2, aux, auxx)
    end;
    add 2(aux, auxx, 1, 0, aux, auxx);
    mul 2(aux, auxx, x, xx, z, zz);

    if xgr1 then s:= s - 9;
    if abs(s) > 1 then
    mul 2(s, 0, pio18, pioo18, pio18, pioo18);
    if s ≠ 0 then add 2(pio18, pioo18, z, zz, z, zz);
    if xneg∧¬xgr1 ∨ xgr1∧¬xneg then begin z:= -z; zz:= -zz end;
    norm(z, zz)
end lng arctan;
```

References

Achieser, N.I.
Vorlesungen über Approximationstheorie.
Akademie-Verlag, Berlin (1953).

Barning, F.J.M.
Standardfunctions in the X8 assembler-code ELAN.
Mathematical Centre, report R1042 (1965) (in Dutch).

Dekker, T.J.
A floating-point technique for extending the available precision.
Numer. Math. 18 (1971) 224.

Fike, C.T.
Computer evaluation of mathematical functions.
Prentice-Hall Inc., Englewood Cliffs, N.J. (1968).

Grau, A.A.
On a floating-point number representation for use with algorithmic languages.
Comm. ACM 5 (1962) 160.

Grune, D.
Manual of the MILLI-system for the EL-X8.
Mathematical Centre, LR 1.1 (1972) (in Dutch).

Kruseman Aretz, F.E.J. and Mailloux, B.J.
The ELAN source text for the MC-ALGOL 60-system for the EL X8.
Mathematical Centre, report MR 84 (1966).

Kruseman Aretz, F.E.J.
Lecture notes, Univ. of Amsterdam.
(unpublished).

Lanczos, C.
Applied analysis.
Prentice-Hall, New York (1956).

Meinardus, G.
Approximation of functions: Theory and numerical methods.
Springer Verlag, New York (1967).

Naur, P. (ed.)
Revised report on the algorithmic language ALGOL 60.
a/s Regnecentralen, Copenhagen (1964).

Remez, E. Ja.
Sur la détermination des polynomes d'approximation de degrée donnée.
Comm. Coc. Math. Kharkov, 10 (1934).

Index to the algorithms