Precise numerical computation☆

Mark Sofroniou a, Giulia Spaletta b, *

a Research and Development, Wolfram Research, Champaign, IL 61820-7237, USA
b Mathematics Department, Bologna University, Piazza di Porta S. Donato, 5, 40127 Bologna, Italy

Dedicated to Jerry B. Keiper (1953–1995) and Nicholas C. Metropolis (1915–1999)

Abstract

Arithmetic systems such as those based on IEEE standards currently make no attempt to track the propagation of errors. A formal error analysis, however, can be complicated and is often confined to the realm of experts in numerical analysis. In recent years, there has been a resurgence of interest in automated methods for accurately monitoring the error propagation. In this article, a floating-point system based on significance arithmetic will be described. Details of the implementation in Mathematica will be given along with examples that illustrate the design goals and differences over conventional fixed-precision floating-point systems.

© 2004 Elsevier Inc. All rights reserved.

AMS classification: 65G20; 65G30; 65G50

Keywords: Significance arithmetic; Interval arithmetic; Error analysis; Accuracy and precision; Condition number; Computer algebra systems; Computer aspects of numerical algorithms

1. Introduction

Knuth has expressed a desiderata of floating-point arithmetic as follows [19]:

It would be nice if we could give our input data for each problem in an unnormalized form which expresses how much precision is assumed, and if the output would indicate just how much precision is known in the answer.

This enhancement would assist those who do not wish to undertake a rigorous analysis of computational error. Significance arithmetic is one approach to obtaining such a facility by providing local error monitoring. A floating-point system based on significance arithmetic dynamically adjusts the number of digits used in computations; it is therefore ideally

☆ Work partially supported by the grants ‘CNR Agency 2000 Computational grids and applications’ and GNCS 2002 ‘Analysis and development of parallel computational kernels for problem solving’.

* Corresponding author.

1567-8326/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jlap.2004.07.007
suited to a software implementation where the mantissa can readily change in size over a very wide range. A premise of Mathematica’s arbitrary precision significance arithmetic is that it is preferable that the dynamic control of digits in low-order operations should be based upon the precision of the operands rather than by considerations such as fixed format storage limitations.

Significance arithmetic is not new and has been around since the late 1950s [8,22,23]. Despite an initial period of enthusiasm, one of the reasons the model fell out of favor was inefficiency compared with alternative hardware implementations of fixed-precision arithmetic models, such as that which evolved into the IEEE 754 standard. Hardware implementations also imposed severe restrictions on the range of representable numbers. Furthermore error estimates, that were obtained using an integral value for the associated error, gave results that were frequently too pessimistic to be of practical value [40].

Recently there has been a resurgence of interest in automatic error control. Nowadays, many modern computational environments are equipped with facilities for carrying out arithmetic in software. Furthermore, there seems to be an increasing demand from the user community for tools which assist in a numerical study of problems, but at the same time give some guarantee of reliability of the solution.

There are some very detailed descriptions of IEEE arithmetic, such as [31], but significance arithmetic is less well known: details of the implementation in Mathematica are not widely available and indeed the possibility of using adaptive precision, with its associated error bounds, offered by this computer algebra system is sometimes overlooked entirely [31, pp. 92–93]. One goal here is to fill this void. A description of how errors are represented and combined in the implementation is given and some enhancements since the early days of development are outlined.

This article is organized as follows. Section 2 begins by introducing some standard notions and the terminology used in Mathematica. The error propagation model used in significance arithmetic is then explained in Section 3. In order to make this article as self-contained as possible, the relevant commands are introduced in Section 4 together with details of the implementation. Section 5 contains examples that are problematic for IEEE arithmetic and these are compared and contrasted with the results obtained using significance arithmetic. Some of the properties of significance arithmetic and functions that can be used to verify the implementation are also outlined. All computations in this article have been carried out using a pre-release version of Mathematica running on a Pentium IV processor machine with RedHat Linux 8.0.

2. Background

In this section, some standard definitions that are used in connection with floating-point arithmetic and error analysis are introduced, the ideas underlying significance arithmetic are described and the terminology that is used in Mathematica is explained.

2.1. Representation of numbers

Several definitions are now recalled to illustrate how real numbers are represented in a computer [20]. A base-$\beta$ finite number $X$ of length $n = k + m$ is an $n$-tuple with radix point between the $k$ most significant digits and the $m$ least significant digits:
The fixed point representation of $X$ is given by:

$$X = x_{k-1} \beta^{k-1} + x_{k-2} \beta^{k-2} + \cdots + x_1 \beta + x_0 + x_{-1} \beta^{-1} + \cdots + x_{-m} \beta^{-m}$$

$$= \sum_{i=m}^{k-1} x_i \beta^i, \quad 0 \leq x_i < \beta \quad \forall i.$$ 

Commonly used positions for the radix point are at the rightmost side of the number (pure integers, $m = 0$) and at the leftmost side of the number (pure fractions, $k = 0$).

The signed-magnitude representation of a floating-point number $X$ in base-$\beta$ consists of three parts, the sign $S$, the (unsigned) mantissa or significant $M$ and the exponent $E$:

$$X = (S, M, E) \beta^E, \quad S \in \{0, 1\}. \quad (1)$$

The mantissa $M \neq 0$ is said to be normalized if $x_{k-1} \neq 0$. A normalized mantissa $M$ can be represented as an integer by taking $\beta^m \leq M < \beta^{m+1}$ or as a fraction by taking $1/\beta \leq M < 1$ and adjusting the exponent accordingly.

A term that will be used occasionally here is the weight $\beta^{-m}$ of the least significant digit, which is commonly referred to as an ulp and signifies a Unit in the Last Place [19] or Unit in the Last Position [20].

Any real number $\alpha$, whose value lies between two consecutive floating-point numbers $X_j = M_j \beta^E$ and $X_{j+1} = (M_j + ulp) \beta^E$, is mapped onto one of these two numbers. A larger distance between $X_j$ and $X_{j+1}$ results in a larger absolute error in the representation, since there is a dependence on the exponent. Upon rounding to the nearest representable number, the maximum absolute representation error is $\beta^E ulp / 2$, and the maximum relative representation error is always bounded as $\beta ulp / 2$ [20].

In IEEE 754 arithmetic the base is taken as $\beta = 2$ and numbers are represented as native types in hardware. In contrast, arbitrary precision numbers are represented in software where $M$ is usually an array of unsigned integers. Commonly used bases are $\beta = 2^{16}$, $\beta = 2^{32}$ or $\beta = 2^{64}$, depending on the architecture and whether hardware integer arithmetic or assembler or software routines are used to multiply and add base-$\beta$ digits.

### 2.2. Normalization of numbers

An example probably best serves to illustrate the ideas involved in significance arithmetic. Assume that two floating-point numbers $X_1 = 0.11112$ and $X_2 = 0.11012$ are given, represented using 4 digit base 2 arithmetic, and that $X_1$ and $X_2$ have no representation error. The subtraction $X_1 - X_2$ yields the four digit number $0.00102$. Normalizing the result then gives the number $0.10002$ with a base 2 exponent of $-2$. The situation becomes more complicated if $X_1$ or $X_2$ themselves contain representation errors or errors from previous computations. A problem with the normalization step is that it discards information relating to how many digits of the result accurately reflect the known digits in the original data: in the example above only two of the four digits in the result are actually justified. One way of overcoming this difficulty is to maintain all numbers in unnormalized form.

---

1 The IEEE 854 standard allows choices other than $\beta = 2$. 
Another is to associate a normalized number with a quantity that keeps track of the scale of the error and hence gives an indication of which digits are considered to be correct. Early implementations of significance arithmetic used an integer indicator to represent the boundary between significant and insignificant digits. This is sometimes referred to as the index of significance [22] or the residence of the least significant digit [26]. The sequel contains a description of how the process of tracking errors is accomplished in Mathematica.

2.3. Scale, accuracy, precision

The terms accuracy and precision are often confused or used interchangeably. In most numerical analysis contexts accuracy refers to the absolute or relative error of an approximate quantity; precision usually refers to the number of digits with which the basic arithmetic operations $+,-,\times,/$ are performed. See [13] for more discussion of this topic.

Quantities that are used to describe errors are given in the decimal system in Mathematica, since this is the default base used for input and output of numbers. This has the advantage that it is irrespective of the base that is actually used to represent numbers internally—a quantity which is platform dependent and also varies between hardware and software numbers.

The decimal scale of a number $x \neq 0$ returned by the Mathematica function \texttt{Scale}, whose implementation is shown in Section 5.4, is defined as:

$$\log_{10}(|x|) \quad (2)$$

Denote by $A_x$ the absolute error of a number $x$ and $A_x/x$ the relative error of $x \neq 0$. In Mathematica’s significance arithmetic, \textbf{Accuracy} is the negative of the scale of the absolute error of $x$:

$$- \log_{10}(|A_x|) \quad (3)$$

\textbf{Precision} is the negative of the scale of the relative error of $x$:

$$- \log_{10}\left(|\frac{A_x}{x}|\right) \quad (4)$$

Of central importance to the implementation of significance arithmetic in Mathematica is the following expression, which can be derived by comparing definitions (2)–(4):

$$\text{Precision}[x] = \text{Scale}[x] + \text{Accuracy}[x], \quad (5)$$

where a Mathematica equation in terms of functions has been used to represent the identity. The value $x = 0$ is an exception in (5) which reflects the fact that the relative error is undefined.

Figs. 1–3 illustrate common situations that occur in the relationship defined by (5) for various values of \textbf{Accuracy}, \textbf{Scale} and \textbf{Precision}. The radix point is assumed to be fixed in absolute position at zero. Note that the values can be negative and that the figures are intended for illustrative purposes only; the values in Mathematica are not necessarily integers, which in effect means that the quantities do not align exactly at the digit boundaries.

\textbf{Accuracy} can be thought of as the approximate number of digits to the right of the decimal (radix) point, \textbf{Scale} as the approximate number of digits to the left of the decimal (radix) point and \textbf{Precision} as the approximate number of significant digits. While these
notions can be reasonable aids to understanding the terminology, they should not obscure
the definitions in terms of absolute and relative errors.

The highlighted words Accuracy and Precision will be used when referring to the
terms in Mathematica in order to distinguish from the standard definitions of accuracy
and precision.

3. Error model

In order to understand in more detail why Accuracy and Precision have been
defined the way they have in Section 2.3, it is useful to consider some examples that com-
bine errors in an absolute and relative fashion. This also helps to illustrate how
errors are represented and propagated through computations in Mathematica’s significance
arithmetic.

3.1. Addition and multiplication

A computational model that rigorously encapsulates errors is interval arithmetic. A real
interval is a non-empty, closed and bounded subset of the real numbers \( \mathbb{R} \):

\[
[x] = [\underline{x}, \bar{x}] = \{ x \in \mathbb{R} \mid \underline{x} \leq x \leq \bar{x} \},
\]

where \( \underline{x} \) and \( \bar{x} \) represent the lower and upper bounds of the interval \([x]\). At each step of
a computation the upper and lower bounds are rounded outwardly so that the interval is
guaranteed to contain the result.

Define two intervals \([\hat{x}]\) and \([\hat{y}]\) centered at the numbers \(x\) and \(y\):

\[
[\hat{x}] = [x - \delta_x, x + \delta_x], \quad [\hat{y}] = [y - \delta_y, y + \delta_y],
\]

(6)

with absolute half-lengths \(\delta_x\) and \(\delta_y\):

\[
\delta_x = \frac{1}{2} |A_x|, \quad \delta_y = \frac{1}{2} |A_y|,
\]

(7)
The quantity $|A_x|$ can be computed in Mathematica as $10^{-\text{Accuracy}[x]}$. The sum of the two intervals defined by (6) and (7) is given by:

$$[\hat{x}] + [\hat{y}] = [(x + y) - (\delta_x + \delta_y), (x + y) + (\delta_x + \delta_y)].$$

(8)

In significance arithmetic, for real numbers $x, y$ with given error \text{Accuracy}[x] and \text{Accuracy}[y], then \text{Accuracy}[x + y] is computed as:

$$-\log\left[10, 10^{-\text{Accuracy}[x]} + 10^{-\text{Accuracy}[y]}\right].$$

(9)

Unfortunately, most errors do not combine according to a simple rule like (8). Instead of (6) now define two intervals:

$$[\hat{x}] = [x(1 - \varepsilon_x), x(1 + \varepsilon_x)], \quad [\hat{y}] = [y(1 - \varepsilon_y), y(1 + \varepsilon_y)],$$

(10)

with relative half-lengths $\varepsilon_x$ and $\varepsilon_y$ for $x, y \neq 0$:

$$\varepsilon_x = \frac{1}{2} \left| \frac{A_x}{x} \right|, \quad \varepsilon_y = \frac{1}{2} \left| \frac{A_y}{y} \right|.$$

(11)

The quantity $|A_x/x|$ can be computed as $10^{-\text{Precision}[x]}$. The product of the two intervals defined by (10) and (11) involves quantities such as:

$$xy(1 + \varepsilon_x + \varepsilon_y + \varepsilon_x\varepsilon_y).$$

(12)

Significance arithmetic in Mathematica neglects error terms that are higher than first order, so that the following leading term approximation is used in place of (12):

$$xy(1 + \varepsilon_x + \varepsilon_y).$$

(13)

Therefore \text{Precision}[x * y] is computed in terms of \text{Precision}[x] and \text{Precision}[y] as:

$$-\log\left[10, 10^{-\text{Precision}[x]} + 10^{-\text{Precision}[y]}\right].$$

(14)

### 3.2. Condition numbers

Some definitions used to estimate the propagation of errors that are related to the conditioning of a function are now recalled. These are established terms in the numerical analysis community, but they serve to highlight the correspondence with the terminology used in Mathematica, explain the behaviour of some examples in the sequel and make the description more accessible to readers from other fields.

Let us examine the result of applying a unary function $f$ to an approximate value $x + A_x$. Assume for simplicity that $f$ is twice continuously differentiable; then it is possible to expand using a Taylor series to obtain:

$$f(x + A_x) - f(x) = f'(x)A_x + O(\Delta^2).$$

Neglecting higher order terms yields the absolute error measure

$$\Delta f(x) = f(x + A_x) - f(x) \approx f'(x)A_x.$$ (15)

Normalizing by dividing both sides in (15) by $f(x)$ gives a relative error measure

$$\frac{\Delta f(x)}{f(x)} \approx \frac{f'(x)}{f(x)} \cdot \frac{A_x}{x}, \quad x, f(x) \neq 0.$$

(16)
The quantity that occurs in (16), namely
\[
c_r(f, x) = \left| \frac{xf'(x)}{f(x)} \right| \tag{17}
\]
is known as the relative condition number of \( f \) (see for example [4,37,13]). The corresponding quantity in (15):
\[
c_a(f, x) = |f'(x)|
\]
is known as the absolute condition number of \( f \). The relative (absolute) condition number measures a relative (absolute) change in an output for a given relative (absolute) change in an input.

Taking absolute values and logarithms in (15) the approximate scale of the error (Accuracy) of a function \( f \) can be defined in terms of the scale of the error of the input. Similarly, taking absolute values and logarithms in (16), the Precision of the input can be related to the Precision of \( f \).

### 3.3. Linearized error model

Numerical algorithms for computing elementary functions can be written in terms of addition and multiplication at some level. However, relying on the error propagation rules for these operations would often give very pessimistic error bounds in significance arithmetic. Much tighter bounds can be obtained by directly imposing error estimates based on properties of functions during their numerical computation.

The error for the exponential function, evaluated at \( x \), propagates such that the Precision of the output is related to the Accuracy of the input:
\[
\frac{A_{\text{exp}(x)}}{\exp(x)} \approx A_x.
\]

For the logarithm function, evaluated at \( x \neq 0 \), the Accuracy of the output is related to the Precision of the input, since the error propagates as:
\[
A_{\text{log}(x)} \approx \frac{A_x}{x}.
\]

The error propagation for certain functions are related to each other; for example, the Accuracy of the sine and cosine functions, evaluated at \( x \), are related to the Accuracy of \( x \) in the following way:
\[
A_{\sin(x)} \approx \cos(x)A_x, \quad A_{\cos(x)} \approx \sin(x)A_x.
\]

As a final example, the error propagation in evaluating powers \( x^y \), \( x \neq 0 \) is:
\[
\frac{A_{x^y}}{x^y} \approx y \frac{A_x}{x}.
\]

The Precision of the input is related to the Precision of the output. Thus the magnitude of the exponent \( y \) determines whether the output has larger or smaller Precision than the input.
4. Syntax and implementation

This section contains worked examples of how Mathematica’s significance arithmetic can be used, as well as a brief description of some design goals.

4.1. Constructors

In order to create a number with Precision 20 any of the following forms can be used (yielding the output 53124.0000000000000000):

- SetPrecision[53124, 20], 53124.20, 5.312420\times10^4, 5.312420\times10^4.

Analogously, a number with Accuracy 20 can be created using any of the following forms (yielding the output 53124.00000000000000000000):

- SetAccuracy[53124, 20], 53124.20, 5.312420\times10^4, 5.312420\times10^4.

Since zero is an exceptional value in the significance arithmetic model, it is not possible to create a zero that only has Precision n:

- In[1] := SetPrecision[0, 5]
  Out[1] = 0

In contrast, it is possible to create a zero with Accuracy n:

- In[2] := SetAccuracy[0, 5]
  Out[2] = 0.10^{-5}

The latter result can be thought of as an interval containing zero, in which none of the digits of the significant of the number representing the center of the interval are known to be correct. Such a number is sometimes referred to as an order of magnitude zero [19].

Another important use of SetAccuracy and SetPrecision is to explicitly override the propagation of errors in a computation. A user may know that, due to the algorithm they are using, rounding errors cancel in some way that is not best represented by an independent linear error model. An example of such a situation is given by Newton’s method in arbitrary precision computations. Under suitable restrictions on the function and the multiplicity of the root, it is well known that errors are strongly correlated and combine in such a way that the number of correct digits approximately doubles at each iteration.

SetAccuracy and SetPrecision enable a shift of the error interval (the boundary between the ‘good’ digits and the ‘bad’ digits) allowing a user to override the default error accumulation model and impose their own alternative. Let us create a Precision 16 software number from the machine floating-point number that represents 0.1:

- In[3] := SetPrecision[0.1, 16]
  Out[3] = 0.100000000000000

The Precision of the previous software number is now increased. If the boundary is moved past the end of the stored digits, more digits are appended. The additional digits are zero in the internal base representation, even though they may appear as nonzero values in the decimal form that is used for output.
In[4] := SetPrecision[%, 50]
Out[4] = 0.10000000000000000555111512312578270211815834045410

4.2. Examining accuracy and precision

For exact numbers, Precision and Accuracy are infinite.

In[5] := {Precision[2], Accuracy[2]}
Out[5] = {∞, ∞}

For a general expression, Accuracy and Precision return the minimum of the values of the constituent subexpressions.

In[6] := Precision[1.50 + f[3/2 + g[2.7 20]]]
Out[6] = 20

The following statement confirms that the Precision of the output is the same as the Precision of the input.

Out[7] = 23.2

There is no monitoring of Precision or Accuracy when machine numbers are involved since the assumption is that if machine numbers are being used then the primary concern is run-time efficiency. Furthermore, machine arithmetic is contagious, so that if a machine number is involved, the result should be a machine number. Adding a machine number to a software number with Precision 20 yields a machine number.

In[8] := 1.2 + 33.2 20
Out[8] = 34.4

In[9] := MachineNumberQ[
Out[9] = True

The previous remarks assume that a result can be represented as a machine number, but machine underflow or overflow can cause numbers to be forced into the realm of software numbers. Here is the maximum representable number in IEEE double precision:

In[10] := maxmachnum = $MaxMachineNumber
Out[10] = 1.79769 10^{308}

The following addition of two machine numbers results in a machine overflow, which is detected and trapped internally by Mathematica and a software number or bigfloat result is returned.

In[12] := MachineNumberQ[\%]
Out[12] = False

4.3. Historical background

The first version of Mathematica used the so called unnormalized method of significance arithmetic, which essentially counted the number of significant digits remaining after leading zeros [8,23]. Prior to version 2.0, the values of Accuracy and Precision were integers that reflected the index of significance (see Section 2.1). In version 2.0 the values were still integers, but the case of combining two numbers with the same Accuracy or Precision was corrected (downwardly) by one unit. In versions after 2.0 the computation of Accuracy and Precision was changed to reflect a continuous error model [15,26] in accordance with formulae (3) and (4). Accuracy and Precision used to return an integer number, which was a reflection of the early implementation. In version 3.0 and onward, facilities were added for returning unrounded values, though these were never formally documented. Since the values of Precision and Accuracy are floating-point numbers, a tolerance is used to compare values (for a discussion of floating-point comparison see [19, Section 4.2.2]). The user-level relational operators such as Equal, Greater, GreaterEqual, also use a tolerance in floating-point comparison.

5. Examples

This section contains several examples that are problematic for fixed-precision floating-point arithmetic. Section 5.1 contains an example that emphasizes the cancellation of rounding errors. Remedies are given along with an example of how the adaptive precision used in significance arithmetic overcomes the difficulties encountered. An example of a commonly used heuristic concerning increasing fixed-precision for verifying results is detailed in Section 5.2. Several types of arithmetic are applied to illustrate differences in the numerical evaluation together with an explanation of the difficulties encountered. In Section 5.3 an adaptive strategy for overcoming cancellation is described. Functions to track the propagation of errors and validate the implementation are outlined in Section 5.4.

5.1. Cancellation error

Consider the following example that turns up in several practical situations [13].

\[
 f(x) = \frac{\exp(x) - 1}{x}. \tag{18}
\]

Define a routine for computing the function given in (18). A special case handles arguments near zero.

\[
\text{In}[13] := f1[x_] := \text{If} \left[ \text{Abs}[x] \leq \$\text{MinMachineNumber}, 1., \frac{\exp(x) - 1}{x} \right].
\]

\[\text{2}\] The examples in this article return unrounded values for Precision and Accuracy which is the standard behaviour in version 5.0.
As the argument \( x \) tends to zero, \( f(x) \) tends to the limiting value 1 (which can be seen by de L’Hôpital’s rule [10]). Using machine double precision however, severe cancellation for small arguments \( x \ll 1 \) causes the result to be progressively more inaccurate.

\[
\text{In}[14] := \text{Table}[f1[1. \cdot 10^{-i}], \{i, 10, 16\}]
\]
\[
\text{Out}[14] = \{1., 1., 1.00009, 0.999201, 0.999201, 1.11022, 0.\}
\]

5.1.1. Remedies

A remedy for machine precision is to rewrite the function defined in (18) by computing both \( \exp \) and \( \log \) as follows [13]:

\[
g(x) = \frac{y - 1}{\log(y)}, \quad y = \exp(x).
\]

(19)

For small \( x \), let \( \hat{y} = \exp(x)(1 + \delta), |\delta| \leq \text{ulp} \). Neither \( (\hat{y} - 1) \) nor \( \log(\hat{y}) \) represent accurate approximations but the errors made in computing \( (\hat{y} - 1) \) and \( \log(\hat{y}) \) almost completely cancel, so that \( (\hat{y} - 1)/\log(\hat{y}) \) is a very good approximation to \( (y - 1)/\log(y) \).

Defining a routine that implements the remedy given in (19), an accurate result is obtained for small machine numbers. The default printing of machine numbers yields only 6 digits, so \text{InputForm} is used to display all the digits.

\[
\text{In}[15] := f2[x_] := \text{With} \left[ \{y = e^x\}, \text{If} \left[ y == 1, y, \frac{y - 1}{\text{Log}[y]} \right] \right]
\]
\[
\text{In}[16] := \text{InputForm}[\text{Table}[f2[1. \cdot 10^{-i}], \{i, 10, 16\}]]
\]
\[
\text{Out}[16] = \{1.0000000000499998, 1.000000000005, 1.00000000000005, 1.000000000000005, 1.000000000000001, 1.0000000000000001, 1.\}
\]

Another remedy worth mentioning is the function \text{expm1} that the C language provides for the accurate computation of \( \exp(x) - 1 \) in situations when the argument \( x \) is near one in magnitude. The onus is clearly on the implementor to take advantage of such facilities when available.

5.1.2. Variable precision arithmetic

Mathematica’s significance arithmetic raises \text{Precision} according to the conditioning of the exponential. Starting with an input value that has \text{Precision} 16, for example, it can be observed that the \text{Precision} of the output increases according to the \text{Scale} of the input.

\[
\text{In}[17] := \text{Exp}[1. \cdot 160^{-10}]
\]
\[
\text{Out}[17] = 1.00000000010000000000050000
\]
\[
\text{In}[18] := \text{Precision}[%]
\]
\[
\text{Out}[18] = 26
\]

After subtraction of an exact unit we are left with a value which still has \text{Precision} 16.

\[
\text{In}[19] := \text{Exp}[1. \cdot 16 10^{-10}] - 1
\]
\[
\text{Out}[19] = 1.000000000050000 10^{-10}
\]
Therefore the evaluation of (18) using significance arithmetic gives a result with the same precision as the input, without the need for reformulation.

\[
\ln(x) := \frac{e^x - 1}{x}
\]

\[
\text{In}[20] := \text{f3}[x] := \text{If}[x == 0, \text{SetAccuracy}[1, \text{Accuracy}[x]], \frac{e^x - 1}{x}]
\]

\[
\text{In}[21] := \text{Table}[\text{f3}[1 \times 10^{-i}], \{i, 10, 16\}]
\]

\[
\text{Out}[21] = \{1.00000000000005000, 1.0000000000000005000, 1.00000000000000005000, 1.000000000000000005000, 1.0000000000000000000000\}
\]

5.2. A polynomial of rump

A common misconception in numerical computation is that it is possible to compute a quantity using, say, single and double precision and if the two results agree to some number of digits, then those digits must be correct. In the case of ill-conditioning then both results may have no correct digits [13]. The following example of Rump [34] serves to illustrate this point.

Define a function of two arguments that can be used to evaluate the polynomial of Rump numerically.

\[
\text{In}[22] := \text{RumpPolynomial}[x_, y_] := \frac{1335}{4} y^6 + x^2 (11 x^2 y^2 - y^6 - 121 y^4 - 2) + \frac{11}{2} y^8 + \frac{x}{2 y}
\]

The polynomial was evaluated by Rump at the values \(x = 77617\) and \(y = 33096\) using single, double and extended precision computations on an S/370 computer. The input data are exactly representable so that there is no representation error—the only source of error occurs during evaluation. The results are summarized in the following table [12]:

<table>
<thead>
<tr>
<th>Singleprecision</th>
<th>Doubleprecision</th>
<th>Extendedprecision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.172603...</td>
<td>1.1726039400531...</td>
<td>1.172603940053178...</td>
</tr>
</tbody>
</table>

Unfortunately, none of these results are correct. In fact, even the sign is wrong!

Using machine arithmetic in Mathematica we get a similarly wrong, but somewhat different, result.

\[
\text{In}[23] := \text{RumpPolynomial}[77617., 33096.]
\]

\[
\text{Out}[23] = -1.18059 \times 10^{21}
\]

The result obtained depends on the scheme used for integer exponentiation and on the order of evaluation of arithmetic operations, since it is well known that floating-point addition and multiplication are not associative (see [19] for a discussion). Results for Rump’s polynomial using IEEE 754 arithmetic are given in [18].

5.2.1. Fixed-precision arithmetic

As an alternative to significance arithmetic and machine arithmetic, Mathematica also allows the use of fixed-precision arithmetic. In contrast to significance arithmetic the latter makes no attempt to track errors.
In order to illustrate how fixed-precision can be used, define a function `FixedPrecisionEvaluate` which locally sets the same value for the system parameters `$MinPrecision` and `$MaxPrecision` that govern the minimum and maximum number of digits that should be used in computations. The `HoldFirst` attribute is specified to ensure that numerical evaluation is not carried out until the two system parameters are locally reset in the scoping construct `Block`.

```mathematica
In[24] := SetAttributes[FixedPrecisionEvaluate, HoldFirst]
In[25] := FixedPrecisionEvaluate[input_, digits_] := Block[{
  $MaxPrecision = digits, $MinPrecision = digits}, input]
```

The following examples illustrate some differences between fixed-precision arithmetic and significance arithmetic. Multiplying two numbers of the same `Precision` gives a result with the same `Precision`.

```mathematica
In[26] := Precision[FixedPrecisionEvaluate[1.`20 * 2.`20, 20]]
Out[26] = 20
```

Exact cancellation in fixed-precision arithmetic yields an exact (machine integer) zero instead of the order of magnitude zeros of significance arithmetic.

```mathematica
In[27] := FixedPrecisionEvaluate[1.`20 - 1.`20, 20]
Out[27] = 0
```

Returning to the example of Rump, the following numerically evaluates the polynomial for a fixed but increasing numbers of digits.

```mathematica
In[28] := Table[
  FixedPrecisionEvaluate[
    RumpPolynomial[SetPrecision[77617, p], SetPrecision[33096, p]], p]. {p, 25}]
Out[28] =
{-6.1019, -3.71019, -3.691019, -3.6891019, -3.68931019, -3.6893491019, -1.02877171018, -8.58993459109, -8.589934590827109, -8.5899345908274109, -8.58993459082740109, -8.589934590827396109, 1.1726039400531786, -0.8273960599468213681, -0.82739605994682136814, -0.827396059946821368141, -0.8273960599468213681412, -0.82739605994682136814117, -0.827396059946821368141165, -0.8273960599468213681411651}
```

Even for a small number of digits results appear to be converging, but as the number of digits used is increased then the numerical contribution from smaller terms in the polynomial start to contribute to the result. Eventually using enough digits convergence to the correct value is obtained.
The values obtained at low precision are perhaps more accurate than might be expected. This is due to the fact that for efficiency the numbers are extended to the internal base-β representation. Results for evaluating Rump’s polynomial using a package developed under the ARITHMOS project, which implements an arbitrary precision extension of IEEE 754 arithmetic, are given in [7]. When the speed of operations is not a primary concern, the package `NumericalMath` can be used to carry out investigations in fixed-precision using alternative rounding modes.

5.2.2. **Interval arithmetic**

The following example creates and displays two intervals with machine double precision bounds. Each interval has one ulp discrepancy in the upper and lower bounds from the input to `Interval`.

In[29] := \textbf{xapprox} = Interval[77617.]

In[30] := InputForm[xapprox]

Out[30] = Interval[{77616.9999999999, 77617.0000000001}]

In[31] := \textbf{yapprox} = Interval[33096.]

In[32] := InputForm[yapprox]

Out[32] = Interval[{33095.9999999999, 33096.0000000001}]

Evaluation of the polynomial of Rump using interval arithmetic results in a very wide interval.

In[33] := \textbf{RumpPolynomial}[xapprox, yapprox]

Out[33] = Interval[{-4.36819 \times 10^{-22}, 4.13207 \times 10^{-22}}]

This provides a warning that there might be a need to rewrite the problem or to use more than double precision in order to get a reasonable answer.

In the following example, interval arithmetic is used with 50 digit precision numbers. The evaluation is performed using fixed-precision computations throughout.

In[34] := FixedPrecisionEvaluate[
    \textbf{RumpPolynomial}[Interval[77617.50], Interval[33096.50]],
    50]

Out[34] = Interval[{-0.82739605994941090378713113623063719368384245823906,
                       -0.827396059954423183249519905472274094562211616220631}]

The resulting interval width is approximately $5.2 \times 10^{-12}$.

5.2.3. **Significance arithmetic**

We now examine the behaviour of significance arithmetic for numerically evaluating the polynomial of Rump. Using input numbers that have \textbf{Precision} 16 (about the same number of digits as double precision), then the result has no correct significant digits. This is reflected by the printed zero for the mantissa, giving an indication that the problem is ill-conditioned.
In[35] := RumpPolynomial[77617.16, 33096.16]  
Out[35] = 0.1023  
The result is about the same magnitude as the result obtained using interval arithmetic with  
the same precision.  
Specifying that each input number should have a Precision of 50 leads to a result that  
only has a Precision of around 12. This means that approximately 38 digits are considered 
to be in error—we shall soon see that this closely mirrors the conditioning of the problem.  

In[36] := rpapprox = RumpPolynomial[77617.50, 33096.50]  
Out[36] = -0.82739605995  
In[37] := Precision[rpapprox]  
Out[37] = 11.815  
Rational arithmetic can be exploited to obtain the exact result to compare against. Sub- 
tracting the approximated value from the exact value, the twelve digits that were displayed 
are then seen to be correct.  

In[38] := rpexact = RumpPolynomial[77617, 33096]  
Out[38] = -5476766192  
In[39] := rpexact - rpapprox  
Out[39] = 0.10 -11  
This result is consistent with the interval result obtained using 50 digit arithmetic. The  
next section illustrates why a loss of approximately 38 digits was observed in this example.  

5.2.4. Error propagation  
The following function can be used to compute the relative condition number (17) of  
a unary continuously differentiable function:  

In[40] := RelativeConditionNumber[f_, x_] := Abs[xD[f, x] / f]  

For a multivariate polynomial \( p \) acting on an \( n \)-tuple \( x = (x_1, \ldots, x_n)^T \), the relative condition number can be defined for \( p(x) \neq 0 \) as:  
\[
c_r(p, x) = \sum_{i=1}^{n} \left| \frac{x_i}{p(x)} \frac{\partial p(x)}{\partial x_i} \right|.
\]

One way of defining a function to evaluate \( c_r(p, x) \) for the polynomial of Rump is as follows.  

In[41] := rpoly = RumpPolynomial[x, y]  
In[42] := rpolycond = RelativeConditionNumber[rpoly, x]  
+ RelativeConditionNumber[rpoly, y]  

The relative condition number is evaluated using exact values. Using approximate  
numbers as inputs may give rise to the same form of difficulty that was experienced in
numerically evaluating the original polynomial. The Mathematica function \( N \) is used to obtain approximate floating-point values after the condition number is evaluated in exact arithmetic.

\[
\text{In}[43] := N[\text{rpolycond}/.\{x \rightarrow 77617, y \rightarrow 33096\}]
\]
\[
\text{Out}[43] = 3.82748 \times 10^{37}
\]

Taking the logarithm to base ten, the result is very close to the loss of 38 digits of Precision that was observed using significance arithmetic.

\[
\text{In}[44] := \text{Log}[10, \%]
\]
\[
\text{Out}[44] = 37.5829
\]

5.3. A polynomial of Kulisch

The following polynomial example of Kulisch is related to the convergents in the continued fraction expansion of the quadratic irrational number \( \sqrt{3} \) (see [19, Section 4.2.2, Exercise 31]).

Define a function for evaluating the polynomial only at real number inputs (the reason for this restriction will become clear at the end of the section).

\[
\text{In}[45] := \text{KulischPolynomial}[x_{\_\_}, y_{\_\_}] := 2y^2 + (9x^4 - y^4)
\]

Now evaluate the polynomial using double precision arithmetic.

\[
\text{In}[46] := \text{KulischPolynomial}[408855776., 708158977.]
\]
\[
\text{Out}[46] = -3.58905 \times 10^{19}
\]

Next define another function for evaluating the polynomial of Kulisch, with the second term represented in factored form, and evaluate it again using double precision arithmetic.

\[
\text{In}[47] := \text{FactoredKulischPolynomial}[x_{\_\_}, y_{\_\_}] := 2y^2 + (3x^2 - y^2)(3x^2 + y^2)
\]
\[
\text{In}[48] := \text{FactoredKulischPolynomial}[408855776., 708158977.]
\]
\[
\text{Out}[48] = 1.00298 \times 10^{18}
\]

The result is completely different from that obtained with the expanded form and the two results do not even share the same sign.

5.3.1. Using significance arithmetic

Evaluating the expanded and factored forms of the polynomial using software numbers with Precision 16 it can be seen that none of the digits are indicated as being correct.

\[
\text{In}[49] := \text{KulischPolynomial}[408855776.\_16, 708158977.\_16]
\]
\[
\text{Out}[49] = 0. \_16
\]
\[
\text{In}[50] := \text{FactoredKulischPolynomial}[408855776.\_16, 708158977.\_16]
\]
\[
\text{Out}[50] = 0. \_16
\]
Section 5.2 illustrated how to estimate the relative condition number for the multivariate polynomial of Rump. For the polynomial of Kulisch, the logarithm of the relative condition number shows that a loss of approximately 36.3 decimal digits should be expected.

5.3.2. Overcoming cancellation

It would be useful to have a way of automatically obtaining a result to within a prescribed error tolerance. This would be advantageous in the absence of a concrete measure for estimating the conditioning of a problem and it would also simplify the process of finding the number of digits that should be used in computations when the conditioning of a problem is known.

Evaluating the polynomial of Kulisch in factored form using exact arithmetic yields the following result—intermediary computations here involve arbitrary precision integers.

\[
\text{In}\{51\} := \text{FactoredKulischPolynomial}[408855776, 708158977]
\]

\[
\text{Out}\{51\} = 1
\]

Unfortunately, using exact arithmetic can be prohibitively expensive in general. It would be useful to have a way of automatically returning a more reliable result using floating-point arithmetic. Using the intrinsic error estimates provided by significance arithmetic, it is possible to use the size of errors in inputs and outputs in an attempt to get an output within a specified error tolerance. The function \text{N} can be used to accomplish this.

Setting the following attribute allows \text{N} to inherit the knowledge that the function \text{KulischPolynomial} takes inexact (floating-point) numbers as inputs and returns an inexact number for the result.

\[
\text{In}\{52\} := \text{SetAttributes}[\text{KulischPolynomial}, \text{NumericFunction}]
\]

In the evaluation that follows, \text{N} converts integer numbers to software floating-point numbers with an initial \text{Precision} and then examines the result. If the \text{Precision} of the result is deficient, linear extrapolation is used to augment the initial \text{Precision} and the computation is then repeated.

\[
\text{In}\{53\} := \text{N}[\text{KulischPolynomial}[408855776, 708158977], 20]
\]

\[
\text{Out}\{53\} = 1.0000000000000000000
\]

The input pattern restrictions \_\text{Real} in the definition of the function \text{KulischPolynomial} ensure that computations are not performed using exact arithmetic. A result for the adaptive modification of precision which minimizes work for a model based on quadratic multiplication is given in [33].

5.4. Informal verification

In high level systems like \text{Mathematica} there is often more than one way of carrying out a computation, so that the system can be used introspectively to validate results. This can be an important aspect in the verification process of software quality assurance. By exploiting the analysis of linearized error propagation for functions, routines can be defined to compute error propagation quantities from their mathematical definitions. Results can then be compared with the built-in functions to examine how closely the implementation matches the theory.
A function Scale to compute the magnitude of a number in decimal in Mathematica can be implemented as follows, where a special case handles arguments near zero (for the definition see Section 2.3).

\[ \text{Scale}[x] := \begin{cases} \text{Accuracy}[x], & \text{if } \text{Abs}[x] \approx 0 \\ \log_{10} \text{Abs}[x], & \text{otherwise} \end{cases} \]

Define two functions which monitor the Accuracy and the Precision of the result of a unary function \( f \) at a given point \( x \).

\[ \text{PropagateAccuracy}[f, x] := \text{Accuracy}[x] - \text{Scale}[f'[x]] \]
\[ \text{PropagatePrecision}[f, x] := \text{PropagateAccuracy}[f, x] + \text{Scale}[f[x]] \]

Next create a real valued number with Precision 20 and invoke PropagateAccuracy for the function \( \text{Exp} \) at the given value.

\[ \text{val} = \text{SetPrecision}[2 \times 10^{-10}, 20] \]
\[ \text{Out}[57] = 2.0000000000000000000 \times 10^{-10} \]
\[ \text{Out}[58] = 29.699 \]

The result of direct computation using the built-in function \( \text{Exp} \) can now be compared against the previous value to check that the implementation correctly follows the linear error model.

\[ \text{Out}[59] = 29.699 \]

Using PropagatePrecision, it can be observed that the Precision of the result of applying \( \text{Sqrt} \) to the number assigned to \( \text{val} \) is augmented from 20 by one bit, or \( \log_{10}(2) \approx 0.301 \) decimal digits.

\[ \text{Out}[60] = 20.301 \]

The reason for the increase in Precision can be understood by considering the (logarithm of the) relative condition number for \( f(x) = \sqrt{x} \). The result is independent of the actual value of \( x \), so that we can use a symbol \( x \) as input.

\[ \text{Out}[61] = \frac{\log_{10}(\text{RelativeConditionNumber}[\text{Sqrt}[x], x])}{\log_{10}(2)} \]

The Precision of the result of the built-in function \( \text{Sqrt} \) matches the expected value obtained with PropagatePrecision.

\[ \text{Out}[62] = 20.301 \]
6. Conclusions

There exist alternative arithmetic models to significance arithmetic whose purpose is
to accurately track the propagation of errors. One approach is interval arithmetic which
we have briefly mentioned and is described in more detail in [11,35]. Although Mathematica
currently only implements real-valued intervals, interval bounds can be specified
to arbitrary precision. For an application of interval arithmetic to verified solutions
for ordinary differential equations see [6]. Another interesting project is MPFR (multiple
precision floating-point reliable) [28]. It has been shown how Mathematica’s implementa-
tion of significance arithmetic varies the number of digits used in computations according
to the conditioning of a function. In contrast, MPFR endeavors to give a fixed-precision
result that is completely accurate, which may involve internally raising the number of digits
used and redoing a computation. Another interesting feature of MPFR is that it provides
several modes for directional rounding. Other recent work in the study of automatic error
analysis is given in [29,32] and a recent description of some certifiable numerical methods
can be found in [1].

The choice of significance arithmetic as the default in Mathematica has not been uni-
versally favorable (see for example [9]) although some of these criticisms relate to early
deficiencies in the implementation. The tools for verifying results described in Section 5.4
are now used as a routine control during development. It is our hope that this will improve
reliability of the implementation. Further criticism, somewhat justifiably, has centered on
the lack of formal documentation explaining how significance has been implemented and
we hope that the description provided here goes some way towards addressing this aspect.

The computational paradigm used in significance arithmetic is not a panacea however.
Knuth goes on to say (see also [3]):

The results of local error monitoring should be combined with a global analysis to
take into account the interdependence which generally exists between the operands
involved in an elaborate computation.

It is an open question as to how generically useful the assumption of independence of
ersors in significance arithmetic is and whether corrective steps to better reflect the cor-
relation of errors can be carried out. Indeed Mathematica uses fixed-precision arithmetic
instead of significance arithmetic in its large scale numerical routines, such as in linear
algebra and the numerical solution of differential equations; the error bounds provided
by these numerical methods are well studied and provide much tighter bounds than those
based on assumptions of independent error accumulation. The issue of data dependency in
interval Gaussian elimination is discussed in [36]. The issue of error correlation in signif-
ificance arithmetic also arises in interval arithmetic, as mentioned in Section 5.2, where it
gives rise to the dependency problem [12,30].

Unlike interval arithmetic, the error bounds in significance arithmetic are approximate.
As long as error intervals are kept sufficiently small, the model can still be used to accu-
rately track errors. There is some inevitable overhead involved in the tracking of errors
that is associated with significance arithmetic. By embedding run time error bounds in the
arithmetic itself, however, the user is freed from maintaining manual error bounds and it
becomes possible to carry out computations much more efficiently than was previously
considered feasible. At least some computational processes fit the assumptions of signifi-
cance arithmetic very well and the following list of examples are provided as evidence of
this.
• Simplification and operations involving exact quantities, such as \( \pi > \frac{22}{7} \). In Mathematica these are often resolved numerically [38]. This would not be possible if some form of error control were not used.
• Inexact computation of Gröbner bases [16,17].
• An algorithm for inexact root isolation for algebraic numbers (Root objects) [41,42] (see also [21]).
• Significance arithmetic is used in a restricted version of Cylindrical Algebraic Decomposition to decide whether a system of inequalities has solutions and to determine approximate sample points [43,44].
• Euclidean division—error control using significance arithmetic for the efficient computation of continued fractions [39].
• Computation of the number of unrestricted partitions of an integer and the number of partitions into distinct parts (the functions \texttt{PartitionsP} and \texttt{PartitionsQ}). Hardy-Ramanujan-Rademacher formulae are used [2] and the error control in significance arithmetic is utilized to return a correctly rounded integer result.
• The package \texttt{NumericalMath Microscope}, useful for verifying the correctness of machine arithmetic libraries in modern compilers.

There is no substitute for traditional methods of numerical analysis, such as forward and backward analysis which provide tangible error estimates. These tools provide the necessary insight that is required in a thorough investigation of a problem (see for example [5,13]). However, we believe that significance arithmetic is a useful addition to the repertoire of available tools for the numerical investigation of problems. In our opinion the model provides an effective aid for users who are not experts in the analysis and construction of numerical methods but are interested in investigating and solving problems, often against the industrial backdrop of pressing deadlines. It is our hope that the examples shown here provide some justification for these claims. Additional information relevant to significance arithmetic can be found in [14,24,25,27,45,46].

Acknowledgements

The description on historical implementation has been expanded from unpublished notes of Jerry Keiper. The authors wish to thank Nicholas Higham for a number of suggestions and for pointing out [7,18] and the referees for constructive comments and for pointing out [30,33,36]. Paul Zimmermann brought the MPFR project to our attention. Jean Buck helped to track down some of the references to significance arithmetic. The authors would also like to thank the organizers of the workshop on Certified Arithmetic held at École Normale Supérieure de Lyon, France, November 15–16, 2000, where a preliminary version of this work was presented.

References

[28] Details of the Multiple Precision Floating-point Reliable project can be found at <http://www.mpfr.org/>.