Mathematical Issues of Mathematica's BigFloat and Our Resolutions in SNAP Package

Kosaku Nagasaka

Kobe University, Japan
nagasaka@main.h.kobe−u.ac.jp

We report the main purpose of the *Mathematica*'s precision and accuracy tracking system (*Mathematica*'s arbitrary precision numbers) according to the official documents and the paper by Sofroniou and Spaletta, differences between these informations and the actual behaviors of *Mathematica*, and the resolutions for this problem in SNAP package. Briefly, this presentation follows the *Mathematica* book, the *Mathematica* guide book for numerics and the paper by Sofroniou and Spaletta [SS05] and claims differences from the actual behaviors from the mathematical point of view (not the numerical analysis way).

■ Introduction

We have been developing SNAP (Symbolic Numeric Algebra for Polynomials) package for *Mathematica* (see the notebook for International *Mathematica* Symposium 2005), providing various functions for symbolic−numeric computations: computing approximate GCDs, bounding numerical zeros of univariate polynomials, testing irreducibilities of multivariate polynomials, computing separation bounds (or irreducibility radii) of bivariate polynomials and so on. One of the purposes of the package is giving an easy−to−use environment for symbolic−numeric computations, especially for people who are not specialists in that subject. The details of the package are given in the later section. In the package, all the polynomials with numerical errors on their coefficients, are treated as polynomial sets, and the author had tried to implement most of functions guarantee that their results are mathematically correct. However, recently, according to discussions with Wolfram Research Inc., its own precision and accuracy tracking system (*Mathematica*'s arbitrary precision numbers) does not guarantee that the results are mathematically accurate so we began to think that the package had to be modified fundamentally. *Mathematica* only guarantees the first order estimation of numerical errors, according to the paper by Sofroniou and Spaletta [SS05] which is the only one we can know the system of precision and accuracy in *Mathematica* except for other official documents including the *Mathematica* guide book by Trott [Trott05]. The details are given in the next section. We note that it is a tracking system so guaranteeing only the first order terms does not mean any defect of *Mathematica*.

We note that this research is partly helped by Grants−in−Aid for Scientific Research, Ministry of Education, Culture, Sports, Science and Technology, JAPAN, #16700016.

■ Official Behaviors

In this section, we review the documented official behaviors of *Mathematica*'s big floating numbers (arbitrary precision numbers) according to the *Mathematica* book, the *Mathematica* guide book for numerics [Trott05] and the academic paper by Sofroniou and Spaletta [SS05].

□ The *Mathematica* Book

Fundamental informations can be found in the section 3.1.4, 3.1.5 and 3.1.6 in the *Mathematica* book. There are two ways operating with floating point numbers, arbitrary precision and machine precision numbers. We focus on the arbitrary precision numbers only since the machine precision numbers do not have significant precision and accuracy informations.

To make sure, we note that any floating point numbers without explicit precision or accuracy specifications are treated as machine precision numbers in *Mathematica* and most of facts argued in the below are not correct for machine precision numbers.

Precision and Accuracy

We can find the following definitions of precision and accuracy in terms of uncertainty δ of the given number x, in the *Mathematica* book.

In general, the *precision* of an approximate real number is the effective number of decimal digits in it which are treated as significant for computations. The *accuracy* is the effective number of these digits which appear to the right of the decimal point. Note that to achieve full consistency in the treatment of numbers, precision and accuracy often have values that do not correspond to integer numbers of digits.

Mathematica is set up so that if a number x has uncertainty δ, then its true value can lie anywhere in an interval of size δ from $x - \delta/2$ to $x + \delta/2$. An approximate number with accuracy a is defined to have uncertainty 10^{-a}, while a non−zero approximate number with precision p is defined to have uncertainty $|x|\,10^{-p}$.

$$\begin{aligned} &\text{Precision}[x] && -\log_{10}(\delta/|x|) \\ &\text{Accuracy}[x] && -\log_{10}(\delta) \end{aligned}$$

Definitions of precision and accuracy in terms of uncertainty.

Hence, we have the following mathematical facts for the *Mathematica*'s arbitrary precision number x and its true value $\bar{x}$ where $p(x)$ and $a(x)$ denote precision and accuracy of x, respectively, if $x \neq 0$. Moreover, we may suppose, expect or think that the mathematical correct value is included in the set after calculations though this article reports that this expectation is not satisfied by *Mathematica* unfortunately.

$$x \in \left\{ \bar{x} \;\middle|\; \frac{|x - \bar{x}|}{|x|} \leq \frac{10^{-p(x)}}{2} \right\} \tag{1}$$

$$x \in \left\{ \bar{x} \;\middle|\; |x - \bar{x}| \leq \frac{10^{-a(x)}}{2} \right\} \tag{2}$$

In *Mathematica*, built−in functions: Precision, Accuracy, SetPrecision and SetAccuracy are designed for operating the above precision and accuracy. We give some examples of these functions.

```
In[4]:= x = SetPrecision[2, 20] (* equivalent to 2`20 *)
Out[4]= 2.0000000000000000000
In[5]:= y = SetAccuracy[2, 20] (* equivalent to 2``20 *)
Out[5]= 2.0000000000000000000
In[6]:= {Precision[x], Accuracy[x]}
Out[6]= {20., 19.699}
In[7]:= {Precision[y], Accuracy[y]}
Out[7]= {20.301, 20.}
```

Error Tracking System

Mathematica's precision and accuracy tracking system is one of important features of *Mathematica*. So, what is the tracking system? For this question, we can find the following paragraph in the section 3.1.5 in the *Mathematica* book.

When you do a computation, *Mathematica* keeps track of which digits in your result could be affected by unknown digits in your input. It sets the precision of your result so that no affected digits are ever included. This procedure ensures that all digits returned by *Mathematica* are correct, whatever the values of the unknown digits may be.

By the above official document, most of users, especially beginners tend to believe that all significant digits computed by *Mathematica* are mathematically correct. However, this is not true and some actual evidences are shown in the later section.

Converting to Arbitrary Precision Numbers

We can not neglect representing errors when we use floating−point numbers. However, the *Mathematica* book does not have any informations about this big problem. For example, we can not represent 0.1 in base 2 with any finite precision since it is a repeating fractional number. For those numbers, the system have to round them up. If we use machine precision numbers, the specification of this rounding is defined as a part of IEEE754 and *Mathematica* also obeys this standard.

□ The *Mathematica* Guide Book and The Academic Paper

We have to read the *Mathematica* guide book and the academic paper if we want to know the mechanism of arbitrary precision numbers more. In the *Mathematica* guide book for numerics, we can find more detailed informations in the section 1.1.1 (Numbers with an Arbitrary Number of Digits). More mathematical informations can be found in the following two academic papers: one is by Sofroniou and Spaletta [SS05] and the other is by Fateman [Fateman92].

Precision and Accuracy

We can find the following argument in the *Mathematica* guide book for numerics. This is the same definition of the *Mathematica* book. However, there is an author's comment (red colored words).

Reverting these definitions, we can say that a number z with accuracy a and precision p will lie with certainty in the interval $(x - 10^{-a}/2,\ x + 10^{-a}/2) =$ $(x - x\,10^{-p}/2,\ x + x\,10^{-p}/2)$ (a distinction between open and closed intervals is not useful for approximative numbers).

A distinction between open and closed intervals is important mathematically. For example, $\left(0,\ 10^{-10}\right)$ and $\left[0,\ 10^{-10}\right]$ are completely different. We can not decide whether 0 is included in the interval or not if we do not know that the interval is closed. Anyway, we have to distinguish between open and closed if we would like to know the results with precision assurance. We discuss this topic in the later section.

For complex numbers, the definition of accuracy is not straight forward (eg. maximum or minimum of real and imaginary parts). We have not found the official definition in the *Mathematica* book, however, we can see the definition in the *Mathematica* guide book as follows.

The accuracy of a complex number $z = x + i\,y$, x and y being real, is defined in the following way: accuracy$(z) = -1/2 \log_{10}\left(10^{-2\,\text{accuracy}(x)} + 10^{-2\,\text{accuracy}(y)}\right)$. The precision of a complex number $z = x + i\,y$, x and y being real, is defined in the following way: precision(z) = accuracy(z)+$\log_{10}(|z|)$. This is the exact equivalent of the corresponding formula for real numbers.

We note that we can find the same definition except for complex numbers, with more mathematical notations in the paper by Sofroniou and Spaletta [SS05]. Those informations are redundant especially after the above quotes from the *Mathematica* book and the guide book. Please check the section 2 in that paper if you would like to see them.

Error Tracking System

Also, we can find the following paragraph that indicates the same fact as in the *Mathematica* book, in the *Mathematica* guide book for numerics.

Mathematica's high−precision arithmetic is based on significance arithmetic. This means that the result of a high−precision calculation is a number (composed of digits) including the knowledge of which of the digits are correct. In `OutputForm` and `StandardForm` only the certified digits are printed. `InputForm` and `FullForm` will display all digits together with the information of how many of them are correct.

By the above official documents, most of users, especially beginners tend to believe that all significant digits computed by *Mathematica* are mathematically correct. However, this is not true and some actual evidences are shown in the later section. In fact, we can find the following argument in the guide book, which indicates that arbitrary precision numbers are computed by only significance arithmetic without accuracy assurance. This is included in a context: how does *Mathematica* checks an identity numerically?

The probability that the identity would be wrong and that accumulated errors in the carried out arithmetic make a wrong identity correct is vanishing small.

One might think that checking identities numerically and mathematical correctness of significant digits are not the same problem hence there are still possibilities that the above arguments in the *Mathematica* book and the guide book are mathematically correct. Unfortunately, the tracking system is not mathematically correct but it is only a reasonable cost−effective significance arithmetic since we can find the following argument in the guide book and the same words in the paper by Sofroniou and Spaletta.

The formula for determining the precision of the output is based on the first−order Taylor series of $f(x)$. This means that this precision model will be accurate for $\frac{\delta x}{x} \ll 1$. For arguments of $f(x)$ with a small precision, *Mathematica*'s high−precision arithmetic might give either too pessimistic or even wrong results.

In the paper by Sofroniou and Spaletta, they wrote an easier example for understanding. Let x and y be enough large positive arbitrary precision numbers with relative errors ϵ_x and ϵ_y, respectively, such that $\epsilon_x = \frac{1}{2}\cdot 10^{-p(x)}$ and $\epsilon_y = \frac{1}{2}\cdot 10^{-p(y)}$. Hence, we have true values $\overline{x}$ and $\overline{y}$ of x and y satisfying $\overline{x} \in [x(1-\epsilon_x),\ x(1+\epsilon_x)]$ and $\overline{y} \in [y(1-\epsilon_y),\ y(1+\epsilon_y)]$, respectively. We think the product of these two intervals. The result of the product is mathematically as follows.

$$\overline{x}\,\overline{y} \in [x\,y\,(1-\epsilon_x-\epsilon_y+\epsilon_x\,\epsilon_y),\ x\,y\,(1+\epsilon_x+\epsilon_y+\epsilon_x\,\epsilon_y)]$$

However, significance arithmetic in *Mathematica* generates the following interval instead of the above. It neglects the vanishing small value $\epsilon_x \epsilon_y$.

$$x\,y \in [x\,y\,(1-\epsilon_x-\epsilon_y),\ x\,y\,(1+\epsilon_x+\epsilon_y)]$$

Although this error model is not wrong or perhaps we should say that it is the reasonable and conventional significance arithmetic, that the official *Mathematica* book does not have the above information is much dangerous. All the users must have the right to know the mechanism of arbitrary precision numbers in *Mathematica*. We have to know the mechanism before using it to know what we get as answers. The *Mathematica* book should not use "correct" digits since most of users think that "correct" means mathematically correct while it is not mathematically correct.

Moreover, Fateman [Fateman92] reported many things about *Mathematica* (maybe version 1.2). In his paper, he introduced the following 4 versions for error tracking systems. It is a kind of evolutional process. We note that the following statement is a summary hence they are not the actual words.

Version 0: just high−precision or multi−precision numbers (there are not any accuracy informations).
Version 1: ordinary significance arithmetic (3.0 means any numbers between 2.95 and 3.04).
Version 2: accuracy and precision are merely bounds on the error introduced in each computation leading up to the present value, without any accuracy assurance.
Version 3: accuracy and precision are merely bounds on the error introduced in each computation leading up to the present value, with accuracy assurance.

At the time when his paper was wrote, the tracking system of *Mathematica* may be something like version 1. According to the *Mathematica* book, we tend to think that the current tracking system is something like version 3, however, according to the *Mathematica* guide book and the paper by Sofroniou and Spaletta, we now can know that it is something only like version 2.

Converting to Arbitrary Precision Numbers

Even if we read the *Mathematica* guide book and the academic paper, we can not find any informations about this big problem. However, in the *Mathematica* guide book, we can find the following useful undocumented function `$NumberBits`.

Floating−point arithmetic with a variable number of digits. This is the method implemented in the current *Mathematica* kernel. Roughly speaking, the main idea is to simulate interval arithmetic by constantly maintaining a “few more digits” than needed (hereafter called guard digits), and to use them to analyze the error.

We can get knowledge of all (base 2) digits (the explicitly shown ones and the hidden ones) of a number in the form of their binary representation by using the undocumented function `$NumberBits`.

`$NumberBits[`*number*`]` gives a list of the form `{`*sign*`, `*listOfCorrectBits*`, `*listOfGuardBits*`, `*base2Exponent*`}` of the approximate real number *approximateRealNumber*.

By this function, as in the later section, we can check the mechanism of the tracking system. Here, we show some examples.

```
In[2]:= NumericalMath`$NumberBits[0.1`10]

Out[2]= {1, {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
          0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1},
         {1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1,
          1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
          0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0}, -3}

In[3]:= NumericalMath`$NumberBits[0.1`20]

Out[3]= {1, {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
          0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1},
         {0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
          0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
          0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0}, -3}

In[4]:= NumericalMath`$NumberBits[0.1`30]

Out[4]= {1, {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0},
         {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
          1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
          0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0}, -3}
```

Other Remarks

In the guide book, we can find various undocumented informations about numerical calculations.

The behavior of `Equal` and `SameQ` with respect to floating point numbers can be influenced by the two functions `Experimental`$EqualTolerance` and `Experimental`$SameQTolerance`. They represent the number of digits such that two numbers are considered to be equal or the same. The current values are as follows.

This indicates the reason of the following examples.

```
In[47]:= 1.00000000`10 == 1.00000001`10

Out[47]= True

In[48]:= 1.00000000`10 === 1.00000001`10

Out[48]= False

In[51]:= 1.00000000`10 === 1.0000000002`10

Out[51]= True
```

Because the two functions `Experimental`$EqualTolerance` and `Experimental`$SameQTolerance` are the following values, as in the guide book. Anyway, we have to set these values 0s for strict mathematical checking of equality and identity although *Mathematica* neglects all the incorrect digits (*Mathematica* thinks they are incorrect) for checking of equality and identity.

```
In[52]:= {Experimental`$EqualTolerance, Experimental`$SameQTolerance}

Out[52]= {2.10721, 0.30103}
```

```
In[53]:= ?Experimental`$EqualTolerance
```

```
$EqualTolerance gives the number of decimal
   digits by which two numbers can disagree and
   still be considered equal according to Equal. More...
```

```
In[54]:= ?Experimental`$SameQTolerance
```

```
$SameQTolerance gives the number of decimal digits
   by which two numbers can disagree and still be
   considered the same according to SameQ. More...
```

□ Remarks

Mathematica's precision and accuracy tracking system is not designed for numerical computations with precision assurance. Hence, results may have wrong precision and accuracy by misadventure, though they are almost always correct. Moreover, there is no mathematical information about converting to arbitrary precision numbers since the system is designed for having both of reasonable computation speed and reasonable correctness, not for mathematical correctness.

We note that we have been trying to get more informations about *Mathematica*'s arbitrary precision numbers for getting more mathematically trustworthy results and certifying the results calculated by our package. After tons of emails from July to November 2005 with Wolfram Research Inc., we got only these informations: 1) there is the academic paper by Sofroniou and Spaletta [SS05] and 2) the *Mathematica* book is the only official document for numerical computations in *Mathematica* . We have needed any useful informations about the *Mathematica*'s error tracking system and its arbitrary precision numbers and we have not gotten them.

■ Actual Behaviors and Problems

In this section, we show some examples that indicate the official documents (including Sofroniou and Spaletta's paper) are not correct or not enough to describe all the system.

□ Precision and Accuracy

In this section, we figure out the insides of precision and accuracy by the function `NumericalMath`$NumberBits`.

At first, we examine how many number of digits in binary form are used in the case of using SetPrecision. In the below output, we can see {{significant digits, hidden guard digits}, specified digits in binary form} for each specified precision *p* where hidden guard digits may not be significant and only used for keeping accurate computations.

```
In[1]:= Table[{Length /@
           NumericalMath`$NumberBits[SetPrecision[2, p]][[{2, 3}]],
         N[Log[2, 10^p - 1]]}, {p, 1, 20}]
```

```
Out[1]= {{{3, 63}, 3.16993}, {{7, 59}, 6.62936},
         {{10, 56}, 9.96434}, {{13, 53}, 13.2876}, {{17, 49}, 16.6096},
         {{20, 46}, 19.9316}, {{23, 43}, 23.2535}, {{27, 39}, 26.5754},
         {{30, 36}, 29.8974}, {{33, 65}, 33.2193}, {{37, 61}, 36.5412},
         {{40, 58}, 39.8631}, {{43, 55}, 43.1851}, {{47, 51}, 46.507},
         {{50, 48}, 49.8289}, {{53, 45}, 53.1508}, {{56, 42}, 56.4728},
         {{60, 38}, 59.7947}, {{63, 35}, 63.1166}, {{66, 64}, 66.4386}}
```

It seems that a number of significant digits is generated by rounding the input precision into the nearest integer. The following four examples follow this hypothesis. The last two examples indicate the rounding mode at the center.

```
In[2]:= {Length /@
          NumericalMath'$NumberBits[SetPrecision[2, 20.63]][[{2, 3}]],
         N[Log[2, 10^20.63 - 1]]}

Out[2]= {{69, 61}, 68.5314}

In[3]:= {Length /@
          NumericalMath'$NumberBits[SetPrecision[2, 20.62]][[{2, 3}]],
         N[Log[2, 10^20.62 - 1]]}

Out[3]= {{68, 62}, 68.4982}

In[4]:= {Length /@ NumericalMath'$NumberBits[
            SetPrecision[2, 20.620554702982714]][[{2, 3}]],
         FullForm[Log[2, 10^20.620554702982714 - 1]]}

Out[4]= {{68, 62}, 68.5'}

In[5]:= {Length /@ NumericalMath'$NumberBits[
            SetPrecision[2, 20.319524707318730]][[{2, 3}]],
         FullForm[Log[2, 10^20.319524707318730 - 1]]}

Out[5]= {{67, 63}, 67.5'}
```

Also, it seems that a number of hidden guard digits is generated as its summation of numbers of significant and hidden digits to be *something* + "a multiple of 32" as follows.

```
In[6]:= Table[Plus @@ (Length /@ NumericalMath'$NumberBits[
              SetPrecision[2, p]][[{2, 3}]]), {p, 1, 50}]

Out[6]= {66, 66, 66, 66, 66, 66, 66, 66, 66, 98, 98, 98, 98, 98, 98,
         98, 98, 98, 98, 130, 130, 130, 130, 130, 130, 130, 130,
         130, 130, 162, 162, 162, 162, 162, 162, 162, 162, 162, 194,
         194, 194, 194, 194, 194, 194, 194, 194, 226, 226}
```

`$NumberBits[`*number*`]` gives a list of the form {*sign*, *listOfCorrectBits*, *listOfGuardBits*, *base2Exponent*}, so we think that a number of digits used for *sign* and *base2Exponent* is just *something* bits. However this is not true since *base2Exponent* uses 32 bits as follows. Therefore, still there are hidden informations in arbitrary precision numbers or just preserved for future uses.

```
In[7]:= NumericalMath'$NumberBits[$MaxNumber]

Out[7]= {1, {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
         {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, 2147483296}

In[8]:= Log[2, 2147483296.]

Out[8]= 31.

In[9]:= Round@Log[2, 3 + 1.]

Out[9]= 2
```

Though we do not know the purpose of something bits, we can expect the following formula.

$$\begin{aligned}\text{hidden guard digits} = {} & \left\lceil \log \text{n} + \frac{1}{2} + (\text{positive tiny number}) \right\rceil + \\ & \text{significant digits} + \\ & (\text{multiplier of } 32)\end{aligned}$$

where n denotes the given number and $[a]$ means the integer nearest to a.

□ Error Tracking System

According to Sofroniou and Spaletta's paper, for enough large positive arbitrary precision numbers x and y such that true values $\overline{x}$ and $\overline{y}$ of x and y satisfying $\overline{x} \in [x(1-\epsilon_x),\ x(1+\epsilon_x)]$ and $\overline{y} \in [y(1-\epsilon_y),\ y(1+\epsilon_y)]$, respectively. When we compute the product of x and y, *Mathematica* generates an arbitrary precision number corresponding to the following interval.

$$x\,y \in [x\,y\,(1-\epsilon_x-\epsilon_y),\ x\,y\,(1+\epsilon_x+\epsilon_y)]$$

We would like to check whether this argument is correct or not. Let x and y be the following two integers.

```
In[10]:= x = 2;
         y = 4;
```

We suppose that accuracy of these two integers is 20 digits in the decimal system. Hence, accordingly, their absolute uncertainties δ_x and δ_y and their relative uncertainties ϵ_x and ϵ_y are defined as follows. We note that precision and accuracy in *Mathematica* denote a range of its uncertainty, however, in this context, δ_x, δ_y, ϵ_x and ϵ_y are satisfying $\overline{x} \in [x(1-\epsilon_x),\ x(1+\epsilon_x)]$, $\overline{y} \in [y(1-\epsilon_y),\ y(1+\epsilon_y)]$, $\overline{x} \in [x-\delta_x,\ x+\delta_x]$ and $\overline{y} \in [y-\delta_y,\ y+\delta_y]$.

```
In[12]:= acc = 20;
         δx = (10^-acc) / 2;
         δy = (10^-acc) / 2;
         ϵx = δx / x;
         ϵy = δy / y;
```

We calculate accuracy of the product of x and y by different three ways. xyA represents the value calculated by the above definition. xyB represents the value calculated by the mathematical way. xyC represents the value by *Mathematica*'s tracking system.

```
In[17]:= xyA = -Log[10, 2 * x * y * (ϵx + ϵy)];
         xyB = -Log[10, 2 * x * y * (ϵx + ϵy - ϵx * ϵy)];
         xyC = Accuracy[SetAccuracy[x, acc] * SetAccuracy[y, acc]];
```

The result is as follows.

```
In[20]:= showmp[n_] := NumberForm[
           N[n, 2 * MachinePrecision], Ceiling[2 * MachinePrecision]]

In[21]:= TableForm[{{"acc", showmp[acc]}, {"xyA", showmp[xyA]},
           {"xyB", showmp[xyB]}, {"xyC", showmp[xyC]}}]

Out[21]//TableForm=
acc   20.00000000000000000000000000000000
xyA   19.22184874961635636749123320202 0
xyB   19.22184874961635636749159511408 9
xyC   19.22184874961636
```

This result is almost same as the definition. However, the last digit of xyC is different from those of xyA and xyB and this would cause critical problems according to circumstances since the actual value xyC is larger than the theoretical value xyA and this means that we can not trust the tracking system if we want to get mathematically correct results even for basic four operations.

The following other example does not cause any critical problems since the actual value xyC is smaller than the theoretical value xyA.

```
In[22]:= acc = 2708664523/130361190;
         δx = (10^-acc) / 2;
         δy = (10^-acc) / 2;
         εx = δx / x;
         εy = δy / y;
         xyA = -Log[10, 2 * x * y * (εx + εy)];
         xyB = -Log[10, 2 * x * y * (εx + εy - εx * εy)];
         xyC = Accuracy[SetAccuracy[x, acc] * SetAccuracy[y, acc]];
         TableForm[{{"acc", showmp[acc]}, {"xyA", showmp[xyA]},
           {"xyB", showmp[xyB]}, {"xyC", showmp[xyC]}}]

         acc     20.778151250383645623363824770240
         xyA     20.000000000000001990855057972261
         xyB     20.000000000000001990855118290939
         xyC     20.
```

□ Converting to Arbitrary Precision Numbers

We can not find any information how to convert numbers to arbitrary precision numbers. Moreover, at least, converting to arbitrary precision numbers has the following problem.

```
In[37]:= FullForm[2`7]

         2.`7.000000000000001

In[38]:= Table[
          {NumberForm[Precision[SetPrecision[2, p]], 32], p}, {p, 1, 64}]

Out[38]= {{1., 1}, {2., 2}, {3., 3}, {4., 4}, {5.000000000000002, 5},
          {5.999999999999999, 6}, {7.000000000000001, 7},
          {7.999999999999999, 8}, {9., 9}, {10., 10}, {11., 11},
          {12., 12}, {13., 13}, {14., 14}, {15., 15}, {16., 16}, {17., 17},
          {18., 18}, {19., 19}, {20., 20}, {21., 21}, {22., 22},
          {23., 23}, {24., 24}, {25., 25}, {26., 26}, {27., 27},
          {28., 28}, {29., 29}, {30., 30}, {31., 31}, {32., 32},
          {33., 33}, {34., 34}, {35., 35}, {36., 36}, {37., 37},
          {38., 38}, {39., 39}, {40., 40}, {41., 41}, {42., 42},
          {43., 43}, {44., 44}, {44.99999999999999, 45}, {46., 46},
          {47., 47}, {48., 48}, {49., 49}, {50., 50}, {51., 51},
          {52., 52}, {53.00000000000001, 53}, {54., 54}, {55., 55},
          {56., 56}, {57., 57}, {58., 58}, {59., 59}, {60., 60},
          {61., 61}, {61.99999999999999, 62}, {63., 63}, {64., 64}}
```

In these cases, we specifies only 5, 7 or 53 significant digits, however, the result number has a little bit larger significant digits. This may violate the given range of uncertainty. Since according to the guide book, `NumericalMath`$NumberBits` gives all of the binary information of the number, we may think that precision is related to the number of significant digits. However, this is not true. There must exist other binary informations on arbitrary precision numbers preserving precision and accuracy or some conversion mechanisms.

```
In[3]:= Table[{Log[10., -1 + 2^Length[NumericalMath`$NumberBits[
          SetPrecision[2, p]][[2]]]], p}, {p, 1, 64}]

Out[3]= {{0.845098, 1}, {2.1038, 2}, {3.00988, 3}, {3.91334, 4},
  {5.11751, 5}, {6.0206, 6}, {6.92369, 7}, {8.12781, 8},
  {9.0309, 9}, {9.93399, 10}, {11.1381, 11}, {12.0412, 12},
  {12.9443, 13}, {14.1484, 14}, {15.0515, 15}, {15.9546, 16},
  {16.8577, 17}, {18.0618, 18}, {18.9649, 19}, {19.868, 20},
  {21.0721, 21}, {21.9752, 22}, {22.8783, 23}, {24.0824, 24},
  {24.9855, 25}, {25.8886, 26}, {27.0927, 27}, {27.9958, 28},
  {28.8989, 29}, {30.103, 30}, {31.0061, 31}, {31.9092, 32},
  {33.1133, 33}, {34.0164, 34}, {34.9195, 35}, {36.1236, 36},
  {37.0267, 37}, {37.9298, 38}, {39.1339, 39}, {40.037, 40},
  {40.9401, 41}, {42.1442, 42}, {43.0473, 43}, {43.9504, 44},
  {44.8535, 45}, {46.0576, 46}, {46.9607, 47}, {47.8638, 48},
  {49.0679, 49}, {49.971, 50}, {50.8741, 51}, {52.0782, 52},
  {52.9813, 53}, {53.8844, 54}, {55.0885, 55}, {55.9916, 56},
  {56.8947, 57}, {58.0988, 58}, {59.0019, 59}, {59.905, 60},
  {61.1091, 61}, {62.0122, 62}, {62.9153, 63}, {64.1194, 64}}
```

□ Remarks

As in the above sections, there are two big problem: 1) where do precision and accuracy come from? 2) how are precision and accuracy modified? even though we have some kind of official documents. These differences between official and actual behaviors might be occurred by the same undocumented reason, rounding modes for example. Anyway, at this time, mathematically or non−mathematically we should not trust error bounds along with arbitrary precision numbers.

■ Our Resolution

Our package guarantees all the results mathematically except some functions. Hence we have to use significance arithmetic with precision assurance. Basically, in our package, all the coefficients of representing polynomials of polynomial sets with error bounds are exact rational numbers while those error bounds are represented by floating point numbers. Therefore, we need an error tracking system for only the following functionalities.

1) Computing with error bounds that are represented by floating point numbers.

2) Conversing the given *Mathematica* expression into the corresponding exact rational expression with an error bound.

3) Conversing the given our package's expression into the corresponding normal *Mathematica* expression.

4) Computing singular values and matrix inverses.

We note that the following resolutions were not used in earlier versions of our package. After and including version 0.5 use the resolutions.

□ Using Rational Interval Arithmetic and Conversion

Since the *Mathematica*'s error tracking system is not trustworthy, we use rational interval arithmetic in our package internally. However, the *Mathematica*'s own interval arithmetic is given only for real numbers not for complex numbers, hence we have to extend *Mathematica*'s interval arithmetic to complex intervals. This workaround using rational intervals guarantees the internal results in our package though it is time−consuming and mostly over−estimated.

Although internal arithmetic is done by rational interval arithmetic, the rest of problems is converting *Mathematica*'s arbitrary precision numbers into rational intervals and vice versa. However, we can not find any complete information on arbitrary precision numbers. `NumericalMath'$NumberBits` and the following useful functions are only introduced in the *Mathematica* guide book.

```
In[31]:= Float2Rational[nd_List] := First[nd] *
           2^Last[nd] * nd[[2]].Table[2^-i, {i, Length[nd[[2]]]}]

In[32]:= FullFloat2Rational[nd_List] :=
          First[nd] * 2^Last[nd] * Join[nd[[2]], nd[[3]]].
            Table[2^-i, {i, Length[nd[[2]]] + Length[nd[[3]]]}]

In[31]:= Float2Rational[n_?NumberQ] :=
          Float2Rational[NumericalMath'$NumberBits[n]]

In[32]:= FullFloat2Rational[n_?NumberQ] :=
          FullFloat2Rational[NumericalMath'$NumberBits[n]]
```

The above two functions give rational numbers corresponding to internal binary expressions of arbitrary precision numbers. Since our package is for *Mathematica*, we have to obey the manner of *Mathematica* hence we trust all the input values from outside of the package. We leave it up to users' judgements that the arbitrary precision numbers are trustworthy or not.

We convert arbitrary floating numbers into rational intervals very carefully even if they are too over–estimated, while converting rational intervals into floating–point numbers is straight forward, as follows.

```
In[169]:= SNAPFloat2Rational[n_] := Interval[Rationalize[n, 0] +
            {-1, 1} * Max[(10^-Interval[Rationalize[Accuracy[n], 0] *
                    (1 + {-1, 1} * 10^-MachinePrecision)]) / 2]]

In[170]:= SNAPRational2Float[n_Interval] :=
           SetAccuracy[(Min[n] + Max[n]) / 2, -Log[10, Abs[Max[n] - Min[n]]]]
```

At least, these functions run against to the following example which was introduced in the former section.

```
In[186]:= x = 2;
          y = 4;
          acc = 20;
          δx = (10^-acc) / 2;
          δy = (10^-acc) / 2;
          ϵx = δx / x;
          ϵy = δy / y;
          xyA = -Log[10, 2 * x * y * (ϵx + ϵy)];
          xyB = -Log[10, 2 * x * y * (ϵx + ϵy - ϵx * ϵy)];
          xyC = Accuracy[SetAccuracy[x, acc] * SetAccuracy[y, acc]];
          xyD =
            Function[-Log[10, Abs[Max[#] - Min[#]]]][SNAPFloat2Rational[
              SetAccuracy[x, acc] * SetAccuracy[y, acc]]];

In[198]:= TableForm[{{"acc", showmp[acc]},
            {"xyA", showmp[xyA]}, {"xyB", showmp[xyB]},
            {"xyC", showmp[xyC]}, {"xyD", showmp[xyD]}}]

          acc      20.00000000000000000000000000000
          xyA      19.22184874961635636749123320202 0
          xyB      19.22184874961635636749159511408 9
          xyC      19.22184874961636
          xyD      19.22184874961635522275511983732 6
```

In our package, error bounds of polynomial sets are represented by *Mathematica*'s arbitrary precision numbers, hence there are still some possibilities that the result error bounds are wrong mathematically though they may be very rare cases. For this problem, we will prepare the option which can enable all computations in our package by interval

arithmetic, since we have not been able to find enough informations about *Mathematica*'s arbitrary precision numbers and interval arithmetic is the only way to guarantee the result at this time.

□ Adding Precision Assurance for Some Built−in Functions

In our package, we have to compute singular values and matrix inverses with precision assurance. However, as in the previous section, we can not trust precision and accuracy of results by the built−in function SingularValueList, Inverse and so on. Fortunately, we have the following well−known corollary bounding errors of singular values (see Corollary 8.6.2 in [GL96]).

For any matrices A and E in $\mathbb{C}^{n\times m}$ and any positive integer $k \le \min\{n, m\}$, we have

$$| \sigma_k (A + E) - \sigma_k (A) | \le \| E \|_2$$

where $\sigma_s(M)$ and $\| M \|_2$ denote the s−th largest singular value and the 2−norm of M, respectively.

By this fact, we can get the smallest singular value and the matrix inverse with precision assurance as follows.

(step 1) compute singular values of M by the built−in function

(step 2) convert these singular values and the elements of associated matrices to rational intervals

(step 3) compute the upper bound of 2−norm of the residual matrix of M

(step 4) compute the rational intervals by the above corollary, and associated matrix inverse or pseudo inverse

For example, we can use the following function instead of the built−in.

```
SingularValueListWithErrorBound[
  inmtx_List /; ListQ[First[inmtx]], ineps_: Automatic] :=
 Module[{eps, mtx, matU, matD, matV, matE, errB, sigma, i},
  If[NumberQ[ineps], eps = ineps, eps = MachinePrecision];
  If[eps === Infinity, eps = MachinePrecision];
  If[eps < MachinePrecision, eps = MachinePrecision];
  mtx = SetPrecision[inmtx, eps];
  {matU, matD, matV} = SingularValueDecomposition[mtx];
  matU = Map[SNAPFloat2Rational, matU, {2}];
  matD = Map[SNAPFloat2Rational, matD, {2}];
  matV = Map[SNAPFloat2Rational, matV, {2}];
  matE = matU.matD.Transpose[Conjugate[matV]] - inmtx;
  errB = Max[Sqrt[Plus @@ Flatten[Abs[matE]^2]]];
  sigma = Table[matD[[i, i]], {i, 1, Length[First[matD]]}];
  sigma = Map[(# + Interval[{-1, 1} * errB]) &, sigma];
  Return[sigma];
 ]
```

We note that the above function does not work in this notebook since this function requires: 1) prepare complex interval arithmetic, 2) extend SNAPFloat2Rational to complex intervals and 3) extend Conjugate to complex intervals. Those requirements are implemented in our package, however, we can not copy them here due to the page limit.

■ References

[Fateman92] Fateman, R. J., A review of Mathematica, J. Symbolic Comput. 13, 1992, 545−579.

[GL96] Golub, G. H. and Loan, C. F. V., Matrix Computations Third Edition, Johns Hopkins Series in the Mathematical Sciences. The Johns Hopkins University Press, Baltimore, 1996.

[KN05] Nagasaka, K., SNAP, International Mathematica Symposium, IMS 2005, 5−8th August 2005.

[SS05] Sofroniou, M. and Spaletta, G., Precise Numerical Computation, Journal of Logic and Algebraic Programming 64(1), 2005, 113−134.

[Trott05] Trott, M., The *Mathematica* Guide Book for Numerics, Springer, 2005.