Reliable computing: numerical and rounding errors

Tomasz Strek
Institute of Applied Mechanics, Poznan University of Technology,
ul. Piotrowo 3, 60-965 Poznan, Poland

Computer representation of real numbers

The arithmetic performed in a machine involves numbers with only a finite number of digits, with the results that many calculations are performed with approximate representations of the actual numbers.

In typical computer, only a relatively small subset of the real number system is used for the representation of all real numbers. This subset contains only rational numbers, both positive and negative, and stores a fractional part, called the mantisa, together with an exponential part, called the characteristic.

Floating point systems are specified by four parameters, Fl(B,p,L,U) , and elements of these systems are specified by three parameters, (s,m,e):

- s is the sign of a floating-point number,
- m is its (unsigned) mantissa, and
- e is its exponent.

Certain values of these parameters are reserved for special values (i.e., to represent objects within the system that do not have interpretations as numbers). Otherwise, the number represented is s*m*B^e, where:

- B is the base of the number system.
- p is the precision. It specifies the maximum number of base-B digits in the mantissa of a floating-point number in the machine. Unspecified trailing digits, if any, are assumed to be 0.
- L is the largest negative (base-B) exponent of a representable number. The smallest positive, normalized, representable number is (1,1,L).
- U is the largest positive (base-B) exponent of a representable number. The largest positive representable number is (1,ddd...d,U), where the mantissa has p d's and d=B-1.

IEEE 754 Standard for binary floating-point arithmetic

In this paper we assume the computer arithmetic to satisfy IEEE 754 binary floating-point arithmetic standard. The IEEE 754 Standard for binary floating-point arithmetic has been widely adopted for use on DSP processors. This standard specifies four floating-point number formats including single- and double-precision. Each format contains three components:

- The Exponent. In the IEEE format, exponent representations are biased. This means a fixed value (the bias) is subtracted from the field to get the true exponent value. For example, if the exponent field is 8 bits, then the numbers 0 through 255 are represented, and there is a bias of 127. Some values of the exponent are reserved for flagging infinity, NaN, and denormalized numbers, so the true exponent values range from -126 to 127. If the exponent length is e, the bias is given by 2e-1-1.

- The Fraction. In general, floating-point numbers can be represented in many different ways by shifting the number to the left or right of the radix point and decreasing or increasing the exponent of the radix by a corresponding amount. To simplify operations on these numbers, they are normalized in the IEEE format. A normalized binary number has a fraction with the form 1.F where F has a fixed size for a given data type. Since the leftmost fraction bit is always a 1, it is unnecessary to store this bit and is therefore implicit (or hidden). Thus, an n-bit fraction stores an n+1-bit number. If the exponent length is e and the word length is w, then the fraction length f = w-e-1. IEEE also supports denormalized numbers.
- The Sign Bit. IEEE floating-point numbers use a sign/magnitude representation where the sign bit is explicitly included in the word. Using this representation, a sign bit of 0 represents a positive number and a sign bit of 1 represents a negative number. Both the fraction and the exponent can be positive or negative, but only the fraction has a sign bit. The sign of the exponent is determined by the bias.

In addition to specifying a floating-point format, the IEEE 754 Standard for binary floating-point arithmetic specifies practices and procedures so that predictable results are produced independent of the hardware platform. Specifically, denormalized numbers, are defined to deal with exceptional arithmetic (underflow and overflow).

Denormalized numbers are used to handle cases of exponent underflow. When the exponent of the result is too small (i.e., a negative exponent with too large a magnitude), the result is denormalized by right-shifting the fraction and leaving the exponent at its minimum value. The use of denormalized numbers is also referred to as gradual underflow. Without denormalized numbers, the gap between the smallest representable nonzero number and zero is much wider than the gap between the smallest representable nonzero number and the next larger number.

Gradual underflow fills that gap and reduces the impact of exponent underflow to a level comparable with roundoff among the normalized numbers. Thus, denormalized numbers provide extended range for small numbers at the expense of precision.

The IEEE 754 single precision floating-point format is a 32-bit word divided into a 1-bit sign indicator s, an 8-bit biased exponent E, and a 23-bit fraction F. The double precision (64-bit) floating-point format consists of a 1-bit sign indicator s, an 11-bit biased exponent E, and a 52-bit fraction F.

The relationship between double precision format and the representation of real numbers is given below.

Number Characterization	**Value**
Normalized: $0 < E < 2047$	$(-1)^s(2^{E-1023})(1.F)$
Denormalized: $E = 0$, $F \neq 0$	$(-1)^s(2^{-1022})(0.F)$
Zero: $E = 0$, $F = 0$	$(-1)^s(0)$
Otherwise	exceptional value

On machines with IEEE floating-point format, the smallest positive normalized floating-point number on a particular computer is 2^(-1022) or about 2.2251e-308. Anything smaller underflows or is an IEEE "denormal." The largest floating-point number representable on a particular computer is one bit less than 2^1024 or about 1.7977e+308. Anything larger overflows.

Maple real numbers format

Maple is a complete mathematical problem-solving environment that supports a wide variety of mathematical operations such as numerical analysis, symbolic algebra, and graphics. Maple' s numeric computation environment supports the following number formats: integer, fraction, floating-point, software floating-point, Maple hardware floating-point and complex

The numeric computation environment in Maple is based on the *IEEE Standard 754 for Binary Floating-Point Arithmetic*, and its subsequent generalization to base 10 machines, known as *IEEE/854*. These standards have natural generalizations to the arbitrary precision computing environment in Maple, and by adopting such natural generalizations the hardware and software numeric computing environments are formed into a cohesive whole. The hardware computation environment simulated within Maple (currently under development) is referred to as the Maple hardware computation environment, to distinguish it from the underlying physical hardware computation environment.

Maple floating-point computation environments differ from the IEEE/754 standard by including the following features:

- the extension to symbolic data;
- the extension to complex numerics;
- square root is not considered a basic operation, and does not signal invalid on negative arguments;
- the *RealToComplex* event is included to handle operations that receive real input and return complex results.

A floating-point number (an object of type float or sfloat) is represented internally in Maple by a pair of integers (the mantissa M and the exponent E). This pair is used to construct the floating-point number M*10^E (base is 10). The number of digits carried in the mantissa for floating-point arithmetic is determined by the Maple environment variable *Digits* (default 10). The Maple *Digits* environment variable controls the number of digits that Maple uses when calculating with software floating-point numbers.

Numerical errors

The fundamental problem with most real-number computations is that their accuracy is not guaranteed. Increasing precision does not prevent this problem. Small errors can accumulate rapidly. Limitations in the representation of numbers can quickly cause completely wrong results.

Consider the example of evaluation of real value of equation

$$f_1 = 333.75 \cdot b^6 + a^2 \left(11 \cdot a^2 b^2 - b^6 - 121 \cdot b^4 - 2\right) + 5.5 \cdot b^8 + 0.5 \frac{a}{b}.$$

For $a = 77617.0$ and $b = 33096.0$, this equation yields $f1 = -1.180591620717411\mathrm{e}+021$ when solved using double precision arithmetic with MATLAB.

Calculations made with floating-point numbers of Maple 6 allow us to get more exact result. As we see the accuracy of result depend on the number of digits that Maple uses when calculating with floating-point numbers.

The most big value of monomial in this equation is the $5.5 \cdot b^8$ for $b = 33096.0$. Using fractionc format of monomial, $\frac{55}{10} \cdot b^8$, and integer format of b, $b = 33096$, we can get exact value of result.

The value is 37-digits integer number $\frac{55}{10} b^8 = 7917111340668961361101134701524942848$. As we see in Table 1 the accuracy of calculation of f_1 is growing up when number of digits, used by Maple during calculation, is greater or equal than 37.

Table 1. The real-valued result of f_1 depends on number of digits

Digits	f_1, $a = 77617.0$, $b = 33096.0$
10	$0.5 \cdot 10^{28}$
20	$-0.99999999999999998827 \cdot 10^{17}$
30	$0.100000011726039400531786318588 \cdot 10^{8}$
35	$-298.82739605994682136814116509547982$
36	$21.1726039400531786318588349045201837$
37	$-0.827396059946821368141165095479816292$
50	- 0.82739605994682136814116509547981629199903311578 44

We can write f_1 in the fraction form

$$f_2 = \frac{33375}{100} \cdot b^6 + a^2 \left(11 \cdot a^2 b^2 - b^6 - 121 \cdot b^4 - 2\right) + \frac{55}{10} \cdot b^8 + \frac{1}{2} \frac{a}{b}.$$

For integer values $a = 77617$ and $b = 33096$, this equation yields $f_2 = \frac{-54767}{66192}$, when solved using Maple.

Rounding errors

The result of any operation on fixed-point number is typically stored in a register that is no longer than the original format of number. When the result is put back into the original format, the extra bits must be disposed of. That is, the result must be rounded. Rounding involves going from high precision to lower precision and precision to lower precision and produces quantization errors and computational noise.

There are four rounding modes available in most of programming languages.

- Round toward zero. The computationally simplest rounding mode is to drop all digits beyond the number required. This mode is referred as rounding toward zero. It results in a number whose magnitude is always less than or equal to the more precise original value. That is, all positive numbers are rounded to smaller positive number, while all negative numbers are rounded to smaller negative numbers.
- Round toward nearest. When rounding toward nearest, the number is rounded to the nearest representable value. This mode has the smallest errors associated with it and these errors are symmetric. As a result, rounding towards nearest is the most useful approach for most applications. This rounding mode is default in most languages.
- Round toward ceilling (positive infinity). When rounding toward ceilling, both positive and negative numbers are rounded toward positive infinity.
- Round toward floor (negative infinity). When rounding toward floor, both positive and negative numbers are rounded toward negative infinity.

After a series of arithmetic operations we may not know the exact answer because limitations in the representation of numbers. Even if every operation in the series is performed twice, once rounding to negative infinity and once rounding to positive infinity, we can not be sure that correct answer belongs to obtained interval. The example below shows

Example 1

Let's consider the problem of solution of real linear system

$$\mathbf{Ax} = \mathbf{b} ,$$

where

$$A_{ij} = \frac{1}{i+j-1} \text{ and } b_i = \sum_{j=1}^{n} j \cdot A_{ij} ,$$

with Gaussian elimination algorithm, where n is the number of equations. The correct solutions for this system of equations are successive natural numbers from 1 to n.

Table 2 and Table 3 presents results of solutions of considered real linear system (with $n = 10$) with rounding to negative and positive infinity, and with rounding to nearest and zero, respectively. All real-valued calculations in solving of this problem were made with MATLAB (MATrix LABoratory).

Listing 1
MATLAB M-file with implementation of algorithm for solution of considered real linear system with selected rounding mode available in MATLAB

```
clear all
format long e
n=10;

%rounding to nearest: setround(0)
%rounding to negative infinity: setround(-1)
%rounding to positive infinity: setround(1)
%rounding to zero: setround(2)
```

```
setround(2)
for i=1:n
  for j=1:n
    A(i,j)=1/(i+j-1);
  end;
  b(i)=sum(A(i,:).*[1:n]);
end;
x=b/A; x1=x'
setround(0)
```

Table 2
Results of solutions of considered real linear system with rounding to negative and positive infinity

Nr	Rounding to negative infinity	Rounding topositive infinity
1	1.000000005310460e+000	1.000000005053460e+000
2	1.999999540862342e+000	1.999999544236507e+000
3	3.000009793624234e+000	3.000010006968410e+000
4	3.999910813494155e+000	3.999906941462206e+000
5	5.000426145579732e+000	5.000451732848140e+000
6	5.998826559116256e+000	5.998740682937394e+000
7	7.001928291899554e+000	7.002089885060843e+000
8	7.998133809985666e+000	7.997961108609275e+000
9	9.000981050565368e+000	9.001079010113854e+000
10	9.999783985701424e+000	9.999761078165776e+000

Table 3
Results of solutions of considered real linear system with rounding to nearest and zero

Nr	Rounding to nearest	Rounding to zero
1	1.000000002292007e+000	1.000000005310462e+000
2	1.999999802158318e+000	1.999999540862339e+000
3	3.000004206701762e+000	3.000009793624240e+000
4	3.999961835015409e+000	3.999910813494132e+000
5	5.000181647215761e+000	5.000426145579755e+000
6	5.999501728244047e+000	5.998826559116256e+000
7	7.000815802077731e+000	7.001928291899554e+000
8	7.999213197295525e+000	7.998133809985666e+000
9	9.000412283324703e+000	9.000981050565368e+000
10	9.999909494163667e+000	9.999783985701424e+000

Example 2 - Equation that causes a rounding error

The limitations of floating-point arithmetic are shown more clearly in this example. Obviously, the exact solution to equation

$$f_3 = 1.0 \cdot 10^{30} + 100.0 - 5.0 \cdot 10^{29} - 5.0 \cdot 10^{29} - 10.0$$

is $f_3 = 90$. However, the second term is lost in either single-precision or double-precision floating point arithmetic, causing $f_3 = -10.0$ to be computed instead. Exchanging the terms in f_3,

$$f_4 = 1.0 \cdot 10^{30} - 5.0 \cdot 10^{29} - 5.0 \cdot 10^{29} - 10.0 + 100.0,$$

can rectify the problem. In most of numeric calculation rectification of rounding error problem is impossible or is not so simple to do.

Digits	f_2
10	-10.0
20	-10.0
28	-10.0
29	90.0
500	90.0

There are four standard rounding modes available in Maple (the *Rounding* environment variable controls the rounding of floating-point results):

- nearest round to the nearest possible value, with ties resolved by rounding to the nearest even
- round towards zero
- infinity round towards positive infinity
- -infinity round towards negative infinity.

Rounding Mode	f_1, $a = 77617.0$, $b = 33096.0$; $Digits = 20$
Nearest	$-0.99999999999999998827 \cdot 10^{17}$
0	$0.50000000000000000117 \cdot 10^{18}$
-infinity ($-\infty$)	$-0.89999999999999999883 \cdot 10^{18}$
Infinity (∞)	$0.50000000000000000118 \cdot 10^{18}$

Rounding Mode	f_1, $a = 77617$, $b = 33096$; $Digits = 20$
Nearest	$0.10000000000000000117 \cdot 10^{18}$
0	$-0.39999999999999999882 \cdot 10^{18}$
-infinity ($-\infty$)	$0.59999999999999999883 \cdot 10^{18}$
Infinity (∞)	$0.20000000000000000118 \cdot 10^{18}$

Rounding Mode	f_1, $a = 77617.0$, $b = 33096.0$; $Digits = 40$
Nearest	- 0.827396059946821368141165095479816291999
0	- 0.827396059946821368141165095479816292000
-infinity ($-\infty$)	- 0.827396059946821368141165095479816292000
Infinity (∞)	- 0.827396059946821368141165095479816291999

References

- IEEE Task P754. ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic. IEEE, New York, 1985.