


Symbolic calculation behind floating-point arithmetic: didactic examples and experiment using CAS

Włodzimierz Wojas¹^[0000-0002-1194-7342] and Jan Krupa¹^[0000-0001-5967-6417]

Warsaw University of Life Sciences (SGGW), Department of Applied Mathematics, ul.
Nowoursynowska 159, 02-776 Warsaw, Poland
wlodzimierz_wojas@sggw.edu.pl

Abstract. Floating-point arithmetic (FP) is taught at universities in the framework of different academic courses, for example: Numerical Analysis, Computer Architecture or Operational Systems. In this paper we present some simple „pathological” and „non-pathological” examples for comparison with symbolic calculations which lie behind calculations in FP with double precision. The CAS programs: Mathematica and wxMaxima are used for calculation. We explain, by making calculations directly from the definition of double precision, why in the presented examples such final results were obtained. We present a didactic experiment for students of the Informatics Faculty of Warsaw University of Life Sciences within the course of Numerical Methods.

Keywords: floating-point arithmetic · symbolic calculation · mathematical didactic · CAS

1 Introduction

The FP [3,4,2,5,6,7,1] is ubiquitous in scientific computing. It is taught at universities in the framework of Numerical Analysis, Computer Architecture, Operational Systems and other courses. A rigorous approach to FP is seldom taught in academic courses and also not enough students, scientists and programmers study numerical analysis in details [3]. In this paper we present some didactic examples of symbolic calculations which lie behind calculations in FP (with double precision [2,1]). These examples were prepared by us for students of Informatics faculty of Warsaw University of Life Sciences and demonstrated to them in the framework of Numerical Methods course. Each operation in FP is performed according to a precise-symbolic algorithm. In spite of the fact that FP is based on symbolic operations, it gives approximate results with some exceptions, e.g.: adding, subtracting and multiplying integers; adding, subtracting, multiplying and dividing negative integer powers of 2. We present in this paper simple examples using Mathematica and wxMaxima where the result of operations depend on the interpretation of the user input data (numbers) by CAS functions (such as Solve, Limit, Det) – as symbolic or approximate. The result may also depend whether these CAS functions use more or less clever algorithms. In the second part of this paper, by making calculations directly from the definition of double precision, we explain why in the examples presented such final results were obtained. The didactic approach presented in this article

is multidisciplinary - it combines theoretical issues in the field of numerical analysis with the use of practical CAS computing tools.

2 Didactic motivations

Sauer's book "Numerical Analysis" [6] presents the basics of calculations in FP with double precision with examples not found in our opinion in other literature. The examples presented in this article are, on the one hand, inspired by the examples in this book, but on the other hand, complement and extend them. Such examples are extremely important in teaching FP arithmetic with double precision. The calculations with double precision presented in this article were exercise tasks for students of Informatics faculty of Warsaw University of Life Sciences within the subject of Numerical Methods. These students were presented with floating point arithmetic, including double precision in the subjects: Computer Architecture and Operating Systems. In the framework of these subjects, students calculated with double precision, e.g.: $1 + 2^{-5}$, $1 + 2^{-52}$, $1 + 3 \cdot 2^{-53} - 1$. Such examples are important in double precision arithmetic, but the calculations presented in our examples are a significant complement to them, deepening the understanding of double precision calculations, although they require more tedious calculations (but still within the reach of students' skills).

3 "Pathological" and "Non - Pathological" example of linear equations system solution

Example 1. ("Pathological")

Consider the following system of linear equations and solve them in Mathematica and Maxima

$$\begin{cases} 3x + y = 0, \\ 0.3x + 0.1y = 1. \end{cases}$$

Example 2. ("Non - Pathological")

Consider the following system of linear equations and solve them in Mathematica and Maxima

$$\begin{cases} 2x + y = 0, \\ 0.2x + 0.1y = 1. \end{cases}$$

Let's us notice that generally in **double precision** we have:

Listing 3.1: Mathematica code:

```

In[1]:=0.1+0.1-0.2
Out[1] =0
In[2]:=0.1+0.1+0.1-0.3
Out[2] =5.551115123125783*10^-17
In[3]:=N[2^(-54),16]
Out[3] =5.551115123125783*10^-17

```

Listing 3.2: Mathematica code:

```

In[4]:=-0.3+0.1+0.1+0.1
Out[4] =2.775557561562891*10^-17 (= 2^-55)

```

It follows that in double precision we have:
 $-0.3 + 0.1 + 0.1 + 0.1 \neq 0.1 + 0.1 + 0.1 - 0.3$ (no additivity).

Let's solve the Example 1. First, we solve it standardly using function Solve in Mathematica and solve in Maxima. Next, we solve it again in Mathematica and Maxima using Cramer's rule.

Listing 3.3: Mathematica code for Example 1:

```

In[1]:=eq1={3 x + y == 0, 0.3 x + 0.1 y == 1};
In[2]:= Solve[eq1, {x, y}]
Out[2] = {}
In[3]:= NSolve[eq1, {x, y}]
Out[3] = {}
In[4]:=M={{3, 1}, {0.3, 0.1}};
In[5]:=Det[M]
Out[5] =5.551115123125783*10^-17
In[6]:={M0 = Det[M],
M1 = {{0, 1}, {0.3, 0.1}}; MatrixForm[M1], M10 = Det[M1],
M2 = {{3, 0}, {0.3, 1}}; MatrixForm[M2], M20 = Det[M2],
r1 = {x -> M10/M0, y -> M20/M0}}
Out[6] =r1={x -> -2.40192*10^16, y ->7.20576*10^16}
In[7]:=eq1 /. r1
Out[7] ={True, True}

```

Listing 3.4: Maxima code for Example 1:

```
(%i1) solve([3*x+1*y=0,0.3*x+0.1*y=1],[x,y]);
rat : replaced 0.3 by 3/10 = 0.3
rat : replaced 0.1 by 1/10 = 0.1
(%o1) []
(%i2) M:matrix([3, 1],[0.3,0.1]);
(M) matrix(
      [3, 1],
      [0.3, 0.1]
)
(%i3) d:determinant(M);
(%o3) 5.551115123125783*10^-17
(%i4) [M1:matrix([0,1],[1,0.1]), M2:matrix([3,0],[0.3,1])];
(%o4) [matrix(
      [0, 1],
      [1, 0.1]
), matrix(
      [3, 0],
      [0.3, 1]
)]
(%i5) [d1:determinant(M1),d2:determinant(M2)];
(%o5) [-1,3]
(%i6) [x0:d1/d,y0:d2/d];
(%o6) [-1.801439850948199*10^16,5.404319552844595*10^16]
(%i7) [xy:matrix([x0],[y0]),b:matrix([0],[1]), M.xy-b];
(%o7) [matrix(
      [-1.801439850948199*10^16],
      [5.404319552844595*10^16]
), matrix(
      [0],
      [1]
), matrix(
      [0.0],
      [0.0]
)]
```

We get the following results. Using functions Solve in Mathematica and solve in Maxima we get that system of equations has no solution. But using Cramer's rule in double precision we get a single solution of the system. Solutions in Mathematica and Maxima are different.

Let's solve the Example 2 in Mathematica and Maxima.

Listing 3.5: Mathematica code for Example 2:

```

In[1]:=eq1={2 x + y == 0, 0.2 x + 0.1 y == 1};
In[2]:= Solve[eq1, {x, y}]
Out[2] = {}
In[3]:= NSolve[eq1, {x, y}]
Out[3] = {}
In[4]:=M={{2, 1}, {0.2, 0.1}};
In[5]:=Det[M]
Out[5] =0

```

Listing 3.6: Maxima code for Example 2:

```

(%i1) solve([2*x+1*y=0,0.2*x+0.1*y=1],[x,y]);
rat : replaced 0.2 by 2/10 = 0.2
rat : replaced 0.1 by 1/10 = 0.1
(%o1) []
(%i2) M:matrix([2, 1],[0.2,0.1]);
(M) matrix(
      [2, 1],
      [0.2, 0.1]
)
(%i3) determinant (M);
(%o3) 0.0

```

We get the same results in Mathematica and Maxima that system of equations has no solution. The determinant of the coefficient matrix is equal to 0 so we cannot use Cramer's rule.

4 “Pathological” and “Non - Pathological” example of limit of a real function

Example 3. (“Pathological”)

Let us calculate the following limit: $\lim_{x \rightarrow 0.1^-} \frac{1}{-3x + 0.3}$

Example 4. (“Non - Pathological”)

Let us calculate the following limit: $\lim_{x \rightarrow 0.1^-} \frac{1}{-2x + 0.2}$

Let's solve the Examples 3 and 4 in Mathematica and Maxima.

Listing 4.1: Mathematica code for Examples 3 and 4:

```

In [5]:=f[x_]=1/(-3 x + 0.3);
In [6]:=Limit[f[x], x -> 0.1, Direction -> 1]
Out[6] =∞
In [7]:=f[0.1]
Out[7]=-1.80144*10^16
In [8]:=g[x_]=1/(-2 x + 0.2)
In [9]:=Limit[g[x], x -> 0.1, Direction -> 1]
Out[9] =∞
In [10]:=g[0.1]
Power::infy: Infinite expression 1/0. encountered. >>
ComplexInfinity

```

Listing 4.2: Maxima code for Examples 3 and 4 :

```

(%i1) f(x):=1/(0.3-3*x);
(%o1) f(x):=1/(0.3-3*x)
(%i2) f(0.1);
(%o2) \-1.801439850948199*10^16
(%i4) limit(f(x),x,0.1);
(%o4) \-1.801439850948199*10^16
(%i5) g(x):=1/(0.2-2*x);
(%o5) g(x):=1/(0.2-2*x)
(%i6) g(0.1);
(%o6) expt: undefined: 0 to a negative exponent. -- an error. To debug this
      try: debugmode(true)
(%i7) limit(g(x),x,0.1);
(%o7) infinity

```

We get in Mathematica and Maxima that limit in Example 3 of function $f(x)$ at point 0.1 is equal to infinity but the value of function $f(x)$ at point 0.1 is finite. But in Example 4 the limit of function $g(x)$ at point 0.1 is equal to infinity and the value of function $g(x)$ at point 0.1 is undefined.

5 Floating-point arithmetic: some basic definitions

In this section, we present a model for computer arithmetic of floating point numbers see [6,3,4,2,5,7,1]. There are several models, but to simplify matters we will choose one particular model and describe it in detail. The model we choose is the so-called IEEE 754 Floating Point Standard.

Rounding errors are inevitable when finite-precision computer memory locations are used to represent real, infinite precision numbers.

The IEEE standard consists of a set of binary representations of real numbers. A floating point number consists of three parts: the sign (+ or -), a mantissa, which contains the string of significant bits, and an exponent. The three parts are stored together in

a single computer word. There are three commonly used levels of precision for floating point numbers: single precision, double precision, and extended precision, also known as long-double precision. The number of bits allocated for each floating point number in the three formats is 32, 64, and 80, respectively. The bits are divided among the parts as follows: all three types of precision work essentially the same way. The form of a

precision	sign	exponent	mantissa
single	1	8	23
double	1	11	52
long double	1	15	64

normalized IEEE floating point number is

$$\pm 1.bbb\dots b \times 2^p,$$

where each of the N b 's is 0 or 1, and p is an $M = 11$ (or 8 or 15) bit binary number representing the exponent. Normalization means that, as shown in, the leading (leftmost) bit must be 1.

When a binary number is stored as a normalized floating point number, it is left-justified, meaning that the leftmost 1 is shifted just to the left of the radix point.

The shift is compensated by a change in the exponent. For example, the decimal number 9, which is 1001 in binary, would be stored as $+1.001 \times 2^3$ because a shift of 3 bits, or multiplication by 2^3 , is necessary to move the leftmost one to the correct position. The number machine epsilon, denoted ϵ_{mach} , is the distance between 1 and the smallest floating point number greater than 1. For the IEEE double precision floating point standard, $\epsilon_{\text{mach}} = 2^{-52}$.

We must truncate the binary representation of a number in some way, and in so doing we necessarily make a small error. The most common method of the truncation is rounding. In base 10, numbers are customarily rounded up if the next digit is 5 or higher, and rounded down otherwise. In binary, this corresponds to rounding up if the bit is 1. Specifically, the important bit in the double precision format is the 53rd bit to the right of the radix point, the first one lying outside of the box.

The default rounding technique, implemented by the IEEE standard, is to add 1 to bit 52 (round up) if bit 53 is 1, and to do nothing (round down) to bit 52 if bit 53 is 0, with one exception: if the bits following bit 52 are 10000..., exactly halfway between up and down, we round up or round down according to which choice makes the final bit 52 equal to 0. (Here we are dealing with the mantissa only, since the sign does not play a role.) Why is there the strange exceptional case? Except for this case, the rule means rounding to the normalized floating point number closest to the original number—hence its name, the Rounding to Nearest Rule. The error made in rounding will be equally likely to be up or down. Therefore, the exceptional case, the case where there are two equally distant floating point numbers to round to, should be decided in a way that doesn't prefer up or down systematically. This is to try to avoid the possibility of an unwanted slow drift in long calculations due simply to a biased rounding. The choice to make the final bit 52 equal to 0 in the case of a tie is somewhat arbitrary, but at least it does not display a preference up or down.

IEEE Rounding to Nearest Rule For double precision, if the 53rd bit to the right of the binary point is 0, then round down (truncate after the 52nd bit). If the 53rd bit is 1, then round up (add 1 to the 52 bit), unless all known bits to the right of the 1 are 0's, in which case 1 is added to bit 52 if and only if bit 52 is 1.

Definition. Denote the IEEE double precision floating point number associated to x , using the Rounding to Nearest Rule, by $\text{fl}(x)$. In computer arithmetic, the real number x is replaced with the string of bits $\text{fl}(x)$.

absolute error (of approximation x by $\text{fl}(x)$) = $|\text{fl}(x) - x|$,

and

relative error (of approximation x by $\text{fl}(x)$) = $|\text{fl}(x) - x|/x$, ($x \neq 0$).

In the IEEE machine arithmetic model, the relative rounding error of $\text{fl}(x)$ is no more than one-half machine epsilon: $|\text{fl}(x) - x| \leq \frac{1}{2} \epsilon_{\text{mach}}$

Example 5. (Double precision representation of 9)

From the above description we can formulate the three steps algorithm which determines the double precision representation of real number. We will illustrate it on the example of number 9.

First step: find enough binary digits of 9: $9 = (1001.000\dots 0)_2$.

Second step: normalise the expansion from the first step:

$$9 = (1.001000\dots 0)_2 \times 2^3.$$

When a binary number is stored as a normalized floating point number, it is leftjustified, meaning that the leftmost 1 is shifted just to the left of the radix point.

The shift is compensated by a change in the exponent. For example, the decimal number 9, which is 1001 in binary, would be stored as $+1.001 \times 2^3$ because a shift of 3 bits, or multiplication by 2^3 , is necessary to move the leftmost one to the correct position.

Third step: Apply IEEE Rounding to Nearest Rule:

$$9 = \underbrace{(1.001000\dots 0)}_{52\text{digits}}_2 \times 2^3.$$

The number machine epsilon, denoted ϵ_{mach} , is the distance between 1 and the smallest floating point number greater than 1. For the IEEE double precision floating point standard, $\epsilon_{\text{mach}} = 2^{-52}$.

In the next sections we will use the three steps algorithm illustrated in example 5.

6 Double precision representation of $\frac{1}{10}$



Fig. 1: First step: find enough binary digits of $\frac{1}{10}$

2^{-4} 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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

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
 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56

Fig. 2: Second step: Normalisation of binary expansion of $\frac{1}{10}$

2^{-4} 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
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

Fig. 3: Third step: Rounding of binary expansion of $\frac{1}{10}$

7 Double precision representation of $\frac{2}{10}$

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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

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
 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56

Fig. 4: First step: find enough binary digits of $\frac{2}{10}$

2^{-3} 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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

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
 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56

Fig. 5: Second step: Normalisation of binary expansion of $\frac{2}{10}$

2^{-3} 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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

Fig. 6: Third step: Rounding of binary expansion of $\frac{2}{10}$

8 Double precision representation of $\frac{1}{10} + \frac{1}{10}$

2^{-4} 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 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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 0
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

Fig. 7: First step: Binary addition of $\frac{1}{10} + \frac{1}{10}$

2^{-3} 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
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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 0
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53

Fig. 8: Second step: Normalisation of binary addition of $\frac{1}{10} + \frac{1}{10}$

2^{-3} 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
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

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
 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52

Fig. 9: Third step: Rounding of binary expansion of $\frac{1}{10} + \frac{1}{10}$

Comparing double precision representations of $\frac{2}{10}$ and $\frac{1}{10} + \frac{1}{10}$ we get:

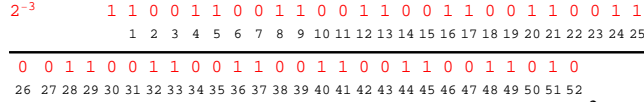


Fig. 10: Third step: double precision representations of $\frac{2}{10}$



Fig. 11: Third step: double precision representations of $\frac{1}{10} + \frac{1}{10}$

Finally, we can see that: $2 * 0.1 - 0.2 = 0$

9 Double precision addition of $\frac{1}{10} + \frac{1}{10} + \frac{1}{10}$

We have already double precision representation of $\frac{1}{10} + \frac{1}{10}$ and $\frac{1}{10}$:



Fig. 12: Double precision representations of $\frac{1}{10} + \frac{1}{10}$



Fig. 13: Double precision representations of $\frac{1}{10}$

and we determine double precision representation of $(\frac{1}{10} + \frac{1}{10}) + \frac{1}{10}$:

“Pre normalisation” of Double precision representation of $\frac{1}{10} + \frac{1}{10}$ and $\frac{1}{10}$:



Fig. 14: “Pre normalisation” of double precision representations of $\frac{1}{10} + \frac{1}{10}$

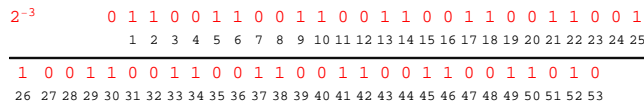


Fig. 15: “Pre normalisation” of double precision representations of $\frac{1}{10}$



Fig. 16: Binary addition of $(\frac{1}{10} + \frac{1}{10}) + \frac{1}{10}$



Fig. 17: Second step: Normalisation of binary addition of $(\frac{1}{10} + \frac{1}{10}) + \frac{1}{10}$



Fig. 18: Rounding of binary expansion of $(\frac{1}{10} + \frac{1}{10}) + \frac{1}{10}$

10 Double precision representation of $\frac{3}{10}$

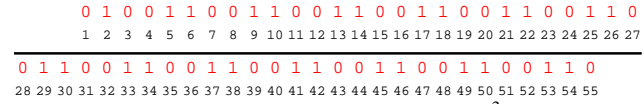


Fig. 19: First step: Binary digits of $\frac{3}{10}$

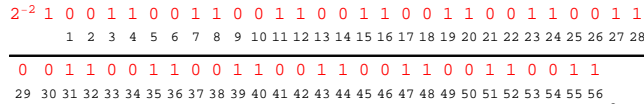


Fig. 20: Second step: Normalisation of binary expansion of $\frac{3}{10}$



Fig. 21: Third step: Rounding of binary expansion of $\frac{3}{10}$

Comparison of double precision addition of $\frac{1}{10} + \frac{1}{10} + \frac{1}{10}$ and of $\frac{3}{10}$

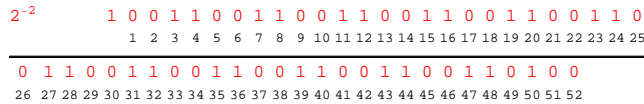


Fig. 22: Third step: Rounding of binary expansion of $\frac{1}{10} + \frac{1}{10} + \frac{1}{10}$



Fig. 23: Third step: Rounding of binary expansion of $\frac{3}{10}$

So, we can see that in double precision: $3 * 0.1 - 0.3 = 2^{-54} = 5.551115123125783 * 10^{-17} \neq 0$.

Similarly we can see that in double precision: $-0.3 + 0.1 + 0.1 + 0.1 = 2^{-55} = 2.775557561562891 * 10^{-17} \neq 0$

We can see also that:

1. The **absolute error** $= |0.1 - \text{fl}(0.1)| = |2^{-4}(2^{-52} + 2^{-56}9 \frac{1}{1-2^{-4}} - 2^{-51})| = 2^{-56} \frac{2}{5} = 2^{-55} \frac{1}{5}$. Thus the **relative error** $= |0.1 - \text{fl}(0.1)|/0.1 = 2^{-54} < (1/2)\epsilon_{\text{mach}} = 2^{-53}$.
Of course there are other good ways to calculate the **absolute error**.
2. See Figs. 10, 11. The **absolute error** $= |0.2 - \text{fl}(0.2)| = 2^{-3}(2^{-52} - 2^{51} + 2^{-56}9 \frac{1}{1-2^{-4}}) = 2^{-55} \frac{2}{5}$. Thus the **relative error** $= |0.2 - \text{fl}(0.2)|/0.2 = 2^{-54} < (1/2)\epsilon_{\text{mach}} = 2^{-53}$.
3. $|0.3 - \text{fl}(0.3)| = |2^{-2}2^{-56}3 \frac{1}{1-2^{-4}}| = 2^{-54} \frac{1}{5}$. Thus $|0.3 - \text{fl}(0.3)|/0.3 = 2^{-54} \frac{2}{3} < (1/2)\epsilon_{\text{mach}} = 2^{-53}$.
4. Because $\text{fl}(0.1 + 0.1) = \text{fl}(0.2)$, thus the **relative error** $= |0.2 - \text{fl}(0.1 + 0.1)|/0.2 = 2^{-54} < (1/2)\epsilon_{\text{mach}} = 2^{-53}$.
5. We saw that $\text{fl}(0.1 + 0.1 + 0.1) - \text{fl}(0.3) = 2^{-54}$ (See Figs. 22, 23). Thus $|0.3 - \text{fl}(0.1 + 0.1 + 0.1)| = |0.3 - \text{fl}(0.3) + \text{fl}(0.3) - \text{fl}(0.1 + 0.1 + 0.1)| = |2^{-54} \frac{1}{5} - 2^{-54}| = 2^{-54} |1/5 - 1| = 2^{-52}/5$. Thus $|0.3 - \text{fl}(0.1 + 0.1 + 0.1)|/0.3 = 2^{-52} 2/3 > 2^{-52} 1/2 = 2^{-53} = (1/2)\epsilon_{\text{mach}} = 2^{-53}$. Thus the **relative error** of approximation (in double precision) 0.3 by $\text{fl}(0.1 + 0.1 + 0.1)$ exceeds $(1/2)\epsilon_{\text{mach}} = 2^{-53}$.
6. Similarly like in Figures 12–18 we can perform $0.3 - 0.1$ in double precision and get: $\text{fl}(0.2) - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1)) = 2^{-55}$. Thus $|0.2 - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1))| = |0.2 - \text{fl}(0.2) + \text{fl}(0.2) - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1))| = \frac{7}{5} 2^{-55}$. Thus $|0.2 - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1))|/0.2 = 7 \cdot 2^{-55} > 2^{-53} = (1/2)\epsilon_{\text{mach}}$. Thus the **relative error** of approximation (in double precision) 0.2 by $\text{fl}(0.3) - \text{fl}(0.1)$ exceeds $(1/2)\epsilon_{\text{mach}} = 2^{-53}$.
7. Similarly like in Figures 12–18 we can perform $0.3 - 0.1 - 0.1$ in double precision and get: $\text{fl}(0.1) - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1)) = 2^{-55}$. Thus $|0.1 - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1))| = |0.1 - \text{fl}(0.1) + \text{fl}(0.1) - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1))| = \frac{1}{5} 2^{-55} + 2^{-55} = \frac{3}{5} 2^{-54}$. Thus $|0.1 - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1))|/0.1 = 3 \cdot 2^{-53} > 2^{-53} = (1/2)\epsilon_{\text{mach}}$. Thus the **relative error** of approximation (in double precision) 0.1 by $\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1)$ exceeds $(1/2)\epsilon_{\text{mach}} = 2^{-53}$.
8. Similarly like in Figures 12–18 we can perform $0.3 - 0.1 - 0.1 - 0.1$ in double precision and get: $\text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1) - \text{fl}(0.1)) = -2^{-55}$. Thus $|0 - \text{fl}(\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1) - \text{fl}(0.1))| = 2^{-55}$. The relative error is not definite.

After the above examples, student can easily see that $1 + 2^{-53} = 1$ in double precision and is prepared to study the very important "pathological" examples in [5,3,4].

11 The didactic experiment with Examples 1-4

The experiment was carried out in two independent auditorium groups (group A and group B) of third-year students of the Informatics Faculty of Warsaw University of Life Sciences within the course of Numerical Methods. Groups A and B consisted of 28 and 31 students, respectively. Students of both groups solved a test consisting of two tasks 1 and 2 with subquestions a) and b) in each task. The time allotted for the test was 30

minutes in each group. But in group B, the test was preceded by a 30-minute presentation in which the double precision numerical representations presented in chapters 6-10 of this article were shown and discussed. Group A students wrote the test without such a prior presentation.

In task 1, two linear systems of equations were presented: in 1a - the system from Example 1 (“pathological”) and in 1b - the system from Example 2 (“non - pathological”) with the comment below. Solving the above systems of equations symbolically (exactly) we get a contradictory system in each case. But when solving them with CAS programs such as Mathematica or Maxima, that is, by calculating the determinants: $\det M$, $\det M_1$ and $\det M_2$ in double precision and then applying Cramer’s formulas, we obtain the following results in cases 1a and 1b. For 1a: $\det M \neq 0$, the system has exactly one solution. For 1b: $\det M = 0$, Cramer’s formulas cannot be used to solve the system in this way. The question is: how, on the basis of properties of double precision floating point arithmetic, can one explain the difference in the answers obtained for linear systems in 1a and 1b?

In task 2, two limits of real function were presented: in 2a - the limit from Example 3 (“pathological”) and in 2b - the limit from Example 4 (“non - pathological”) with the following comment. When solving these limits symbolically we get an infinity in each case. But when solving them in double precision for example using Maxima, we obtain the following results: a finite number in case 2a and infinity in case 2b. The question is: how, based on properties of double precision floating point arithmetic, can one explain the difference in the answers obtained in 2a and 2b?

For tasks 1 and 2, we gave a hint: consider in double precision arithmetic the representations of the following numbers: $0.3, 3 * 0.1$ and $0.2, 2 * 0.1$. Since many students provided quite extensive descriptive explanations, we adopted a two-stage grading scale: positive (if the explanation was sufficient) and negative (if the explanation was not sufficient). We got the following test results. In group A - task 1: 30% positive, 70% negative; task 2: 39% positive, 61% negative. In group B, the results were identical in both tasks: 61% positive, 39% negative. We received a higher percentage of positive answers to both test questions in group B in which the test was preceded by a presentation.

12 Conclusions

In this paper we presented some simple “pathological” and “non-pathological” examples for comparison, using Mathematica and wxMaxima with double precision calculation. Simple systems of linear equations and limits were considered. We used standard CAS functions for solve the systems of linear equations and to calculate limits and determinants. Comparison of results for analogous “pathological” and “non-pathological” examples may seem surprising. In the second part of the paper, by analyzing double precision representations of used in examples numbers, the authors explain why the such final results were obtained. We showed that in double precision: $0.1 + 0.1 - 0.2 = 0$, $0.1 + 0.1 + 0.1 - 0.3 = 2^{-54}$ and explained briefly the way how to calculate that: $-0.3 + 0.1 + 0.1 + 0.1 = 2^{-55}$ and $0.3 - 0.1 - 0.1 - 0.1 = 2^{-55}$. The calculation could be done by hand but we think that it is good idea to implement own function in e.g. Mathematica and wxMaxima to perform this calculation. From the ob-

tained results it follows the main reason of "pathological" character of the presented Examples 1 and 3. In our opinion these examples seems to be helpful to get the basic understanding of the nature of calculation in double precision generally an particularly in the case of used CAS programs. The examples are simple enough to do the calculations by hand but the symbolic calculations behind them are not easy for beginner students in our opinion. We did not find any analogical examples in available literature, taking into account the details we provide. Especially we could not find the calculation of **absolute error** and **relative error** of approximations: 0.1 by $\text{fl}(0.1)$, 0.2 by $\text{fl}(0.2)$, 0.3 by $\text{fl}(0.3)$, 0.2 by $\text{fl}(0.1 + 0.1)$, 0.3 by $\text{fl}(0.1 + 0.1 + 0.1)$, 0.2 by $\text{fl}(0.3) - \text{fl}(0.1)$, 0.1 by $\text{fl}(0.3) - \text{fl}(0.1) - \text{fl}(0.1)$ and comparison it to $(1/2)\epsilon_{\text{mach}}$. We would recommend discussing similar numeric examples in teaching the FP within some subjects in the framework of academic courses. The Examples 1-4 were prepared for students of Informatics Faculty of Warsaw University of Life Sciences. We presented in this article the results of a didactic experiment: students in two independent groups answered test questions related to Examples 1-4. In one of the two student groups, the test was preceded by a presentation discussing numerical representations in double precision arithmetic (chapters 6-10 of this article). We received a higher percentage of positive answers to both test questions in group in which the test was preceded by a presentation. It is worth emphasizing the multidisciplinary nature of the presented didactic approach - combining a theoretical problem in the field of numerical analysis with the use of practical CAS computing tools.

References

1. The gnu mpfr (multiple-precision floating-point computations) library: <https://www.mpfr.org>
2. Ieee 754 floating point standard: https://en.wikipedia.org/wiki/IEEE_754
3. Arnold, J.: An introduction to floating-point arithmetic and computation. <https://indico.cern.ch/event/626147/attachments/1456066/2247140/FloatingPoint.Handout.pdf> (2017)
4. Goldberg, D.: What every computer scientist should know about floating point arithmetic. *ACM Computing Surveys* **23**, 5–48 (1991)
5. Muller, J.M.: *Handbook of Floating-Point Arithmetic*. Birkhäuser, Boston (2010)
6. Sauer, T.: *Numerical Analysis*. Pearson, 3rd edn. (2017)
7. Stallings, W.: *Computer Organization and Architecture*. Prentice Hall, Upper Saddle River, NJ (2003)