How to cite this paper:

Risse, Th. (2018). Towards a mathematical library for UNUMs, an alternative to IEEE 754 floating point numbers. *Journal of Information and Communication Technology* (JICT), *17* (1), 1-16.

TOWARDS A MATHEMATICAL LIBRARY FOR UNUMs, AN ALTERNATIVE TO IEEE 754 FLOATING POINT NUMBERS

Thomas Risse

Institute of Informatics and Automation, Faculty EEE & CS
City University of Applied Sciences
Bremen, Germany

risse@hs-bremen.de

ABSTRACT

The 1985 IEEE 754 standard for the representation of and the arithmetic with floating point numbers has been reconsidered. On the one hand today, its technological assumptions are by no means longer valid. On the other hand, the irritating numerical phenomena which have been collected cast a doubt as to whether this much uncertainty in numerical results is fate. Fortunately, around 2015, Gustafson proposed UNUMs, a modification of the IEEE 754 standard with the potential to heal the said shortcomings. Till now, there are some attempts to implement his ideas, both in software and in hardware. With these activities well under way, the other necessity is development of a mathematical library for UNUMs when one wants UNUMs to become the new floating point standard. This paper presented the ideas leading to UNUMs, gave some hints on floating point units for UNUMs and illustrated the difficulties in developing the said mathematical library by the example of approximating zeroes of analytic functions.

Keywords: Floating point numbers, IEEE 754, UNUMs, arithmetic, mathematical library.

Received: 25 June 2017 Accepted: 3 August 2017

TOWARDS A MATHEMATICAL LIBRARY FOR UNUMs, AN ALTERNATIVE TO IEEE 754 FLOATING POINT NUMBERS

Thomas Risse
Institute of Informatics and Automation, Faculty EEE & CS
City University of Applied Sciences
Bremen, Germany

risse@hs-bremen.de

ABSTRACT

The 1985 IEEE 754 standard for the representation of and the arithmetic with floating point numbers has been reconsidered. On the one hand today, its technological assumptions are by no means longer valid. On the other hand, the irritating numerical phenomena which have been collected cast a doubt as to whether this much uncertainty in numerical results is fate. Fortunately, around 2015, Gustafson proposed UNUMs, a modification of the IEEE 754 standard with the potential to heal the said shortcomings. Till now, there are some attempts to implement his ideas, both in software and in hardware. With these activities well under way, the other necessity is development of a mathematical library for UNUMs when one wants UNUMs to become the new floating point standard. This paper presented the ideas leading to UNUMs, gave some hints on floating point units for UNUMs and illustrated the difficulties in developing the said mathematical library by the example of approximating zeroes of analytic functions.

Keywords: Floating point numbers, IEEE 754, UNUMs, arithmetic, mathematical library.

Received: 25 June 2017 **Accepted**: 3 August 2017

INTRODUCTION

The IEEE 754 standard is to be considered as a milestone in the attempt to make floating number representations and arithmetic machine independent. Before 1985, floating point operations on different machines produced different answers. Against considerable special interests of the industry, IEEE 754 standardizes how floating point numbers are represented in memory and how floating point operations are to be performed (IEEE, 1985 and 2008). Alas, IEEE 754 is a child of its time. It mirrors the technological conditions of the 1980s. And over time a lot of deficiencies have been identified which nevertheless are widely considered as fate. Remedies often worked only at a symptom level (Bailey, 2012).

In 2015 John Gustafson proposed *Universal Numbers, (UNUMs)*, a modification of IEEE 754. These novel formats for floating point numbers have the potential to overcome the shortcomings of IEEE 754 floats and doubles.

Here, we set out to sketch IEEE 754 representations, the arithmetic and its abnormalities. Then we will show how UNUMs promise to remedy both the technological as well as the numerical deficiencies (Gustafson, 2015).

In order to spread the benefits of UNUMs for any kind of scientific computing, a library with the classical numerical methods is indispensable. We present zero finding algorithms and discuss their suitability for UNUMs.

IEEE 754 FLOATING POINT NUMBERS

In general, IEEE 754 floating point numbers are represented by a fixed number of bytes which contain a *sign* bit, a *mantissa* or *fraction* and a power of two given by the *exponent* (the order actually is sign, exponent and mantissa). So IEEE 754 allows the representation of some rational numbers. In order to avoid ambiguities like $1 = 1 \cdot 2^0 = 2 \cdot 2^{-1}$ the mantissa is normalized to lie in the interval [1, 2). Hence, the mantissa has the form $1.\, m_1 m_2 m_3 ...$ with N_{frac} mantissa bits m_1, m_2, m_3 ... so that the first bit need not be stored, it is hidden. Also, the exponent is stored in bits with an offset in order to avoid another sign bit. IEEE 754 specifies most prominently single and double precision floats by

	N_{frac}	N_{expt}	Range	N_{decd}
Floats	24	8	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	≈7
Doubles	53	11	$\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$	≈16

where the number N_{decd} of decimal digits approximates the accuracy achieved. Here, we do not want to dwell on *subnormals* which for the minimal exponent allow not normalized fractions, but we want to point out that there are two representations of ± 0, representations of $+\infty$ as well as $-\infty$ and very many representations of NaN (Not a Number,) which stand for any indefinite expressions like $\infty - \infty$, $\frac{0}{0}$, or $\frac{\infty}{\infty}$. Details are found in the standard (IEEE, 1985 and amended 2008) and in Goldberg (1991).

Now, arithmetic with IEEE 754 is in several aspects by no means trustworthy. Numbers like $0.1 = \frac{1}{10}$ cannot be represented exactly; the laws of associativity or distributivity cannot be guaranteed. There is no indication whether an overflow or an underflow occurred, i.e. whether some numbers bigger than maxreal or smaller than –maxreal is represented by Inf or –Inf respectively or whether a non-zero numbers very close to 0 is represented by 0.0. The only indication may be some flags in the processor status word. Let us list some more very unsettling phenomena in more detail.

- $\frac{99}{7} - \frac{97}{7} - \frac{2}{7} = 9.992007221626409 \cdot 10^{-16}$ in double precision which is close to but not equal to zero.
- Even innocent numbers like $0.1 = \frac{1}{10}$ cannot be represented exactly because $0.1_{(10)} = 0.0\overline{0011}_{(2)}$ has a periodic binary representation.
- For example $(10^{16} - 10^{16}) + 1 = 1 \neq 0 = 10^{16} + (-10^{16} + 1)$ so that associativity cannot be guaranteed. This prevents automatic parallelization of expressions like $a + b + c + d = (a + b) + (c + d)$ by the compiler, different functional floating point units in the processor may not do the two additions $a + b$ and $c + d$ in parallel.
- For example $a + b == a$ not necessarily implies $b == 0$ but $|b| \ll |a|$ only.
- Computing $\sqrt[2^n]{2}$ by $\sqrt{\sqrt{\cdots\sqrt{2}}}$, i.e. taking n times square roots of square roots will eventually lead to the obviously wrong result $\sqrt[2^n]{2} = 1$.
- Solving ill conditioned systems of linear equations naively, i.e. without taking numerical peculiarities into consideration, i.e. here without (partial) pivotization, leads to obviously wrong results. E.g. solving [2]

  $\begin{matrix} 0.25510582\,x + 0.52746197\,y = 0.79981812 \\ 0.80143857\,x + 1.65707065\,y = 2.51270273 \end{matrix}$ by Cramer's rule results in $\binom{x}{y} = \binom{0}{1.3}$

  instead of the exact solution $\binom{x}{y} = \binom{-1}{2}$.
- Evaluate $H(x) := E(Q^2(x))$ with $E(0) := 1$ and $E(x) := \frac{e^x - 1}{x}$ and $Q(x) := |x - \sqrt{x^2+1}| - \frac{1}{|x + \sqrt{x^2+1}|}$ for $x = (15,16,17,9999)$. Then e.g. MATLAB (like any other computing environment) returns $y = (0,0,0,0)$ instead of the correct result $y = (1,1,1,1)$ because actually $Q(x) \equiv 0$.

- Consider the Kahan sequence recursively specified by $u_o = 2, u_1 = -4$ and $u_{i+2} = 111 - \frac{113}{u_{i+1}} + \frac{3000}{u_i\, u_{i+1}}$ which should converge to the stable fix point 6 but instead goes astray to the other fix point 100.
- More examples like those of Kahan can be found in Risse (2016).

There are lists of disasters which occurred in the real world due to these floating point anomalies (Bailey, 2015).

In order to counter these numerical errors one tends to use as much accuracy as available (Bailey, 2014). For example, MATLABs standard data type is double, and thus is independent of the actual requirements.

We conclude this short introduction of IEEE 754 floating point numbers with a comparison of the technological conditions of the 1980s with the conditions forty years later. In the eighties of the last century the 80x86 architecture was predominant with its eight floating point registers, organized as a stack, and performing arithmetic using these registers which were internally 80 bit wide. The difference becomes apparent if one, for example, does floating point arithmetic using JAVA. Also the proportion of time and energy spent for floating point arithmetic compared to moving data between processor and memory has dramatically changed. In the 1980s, when the IEEE standard was finally approved, the following Table 1 (https://en.wikipedia.org/wiki/Intel_8086) allows one to estimate that a floating point operation took hundreds of clock cycles at 0.1μsec each where the memory access with about 10 cycles accounts for less than 10%.

Table 1

Execution Times for Typical Instructions (in clock cycles) at a Clock Rate of 5MHz to 10MHz for 8086

Instruction	Register-register	Register immediate	Register-memory	Memory-register	Memory-immediate
mov	2	4	8+EA	9+EA	10+EA
ALU	3	4	9+EA,	16+EA,	17+EA
jump	11 for register; 15 for label; 16 for condition and label				
integer multiply	70~160 (depending on operand data as well as size) including any EA				
integer divide	80~190 (depending on operand data as well as size) including any EA				

Note: EA = time to compute effective address, ranging from 5 to 12 cycles.
Timings are best case, depending on prefet status, instruction alignment and other factors.

Today, the relation of the execution time of arithmetic operations compared to that of memory access is inverted as the following Table 2 shows (Gustafson, 2016a) with less than 10%. An arithmetic operation takes a fractional amount of the time a memory access takes. The relation in terms of energy consumption is even more stunning and of great importance in today's densely packed high performance multiprocessor systems.

Table 2

Energy and Execution Times for Typical 80x86 Operations

Operation	Energy/pJ	Time/nsec
64 bit multiply add	200	1
Read 64 bit from cache	800	3
move 64 bit across chip	2000	5
Execute an instruction	7500	1
Read 64 bit from DRAM	12000	70

So IEEE 754 has two shortcomings: floating point operations are not trustworthy and the available bandwidth of the channels between processor (or caches) and memory is not used efficiently. Both problems are addressed by UNUMs.

UNIVERSAL NUMBERS

Universal numbers, UNUMs for short, are an extension of IEEE 754 with the potential to solve both the numerical problems of IEEE 754 floating point numbers as well as their inefficiency in exploiting the bandwidth when exchanging floating point numbers between processor and memory and when storing them in memory (Gustafson, 2015). As IEEE 754 numbers UNUMs consist of three fields: the sign bit, the exponent field and the fraction field. The first extension is the fourth field, the so called *ubit* which addresses the numerical problems. If the ubit is not set the number $x = sign * fraction * 2^{exponent}$ is the exact representation of some rational number. If the ubit is set then the UNUM stands for the open real interval $(x, x') \subset \mathbb{R}$ where x' denotes the next exactly representable UNUM. Thus UNUMs cover the whole real line by monotonically alternating between exactly represented rational numbers and open real intervals. Hence UNUMs are either exactly represented rational numbers or open real intervals. As in IEEE 754 also $-\infty$ and $+\infty$ can be expressed as UNUMs as well as quietly and signaling NaN.

With UNUMs there is no rounding and thus no cheating. UNUMs are truly honest about what they represent.

The second extension consists of the two additional fields *esizesize* and *fsizesize* which specify the number of bits of the exponent field and the number of bits of the fraction field, respectively. The two fields render the UNUM format self-descriptive. These two parameters define a so called *environment*.

Now the range and the accuracy of UNUMs can be controlled (in the best case by the computer hardware itself) and thus adapted to the actual needs. The investment in the two fields by far pays out in terms of net memory savings and transmission savings. So UNUMs are the natural candidates for scientific computing on today's high performance computing systems.

Arithmetic with UNUMs

The ubit implies that the objects of UNUM arithmetic are intervals, the so called *ubounds*, namely either closed intervals of the form $[x, x]$ or open and closed intervals $[x, y]$, $[x, y)$, $(x, y]$, or (x, y). Hence for the implementation of each of the basic arithmetic operations two tables are needed which specify the lower bound *low* and the upper bound *upp* of the interval representing the result as is shown by the two addition, ⊕, tables.

Arithmetic with UNUMs never generate an underflow or an overflow. For example, if $x + y >$ maxreal or $x + y < -$ maxreal then the result is the correct ubound (maxreal ∞,) or (– ∞, maxreal), respectively, where maxreal denotes the greatest representable UNUM.

$\oplus_{low}$	$[-\infty$	$(-\infty$	$[y$	(y	$[\infty$
$[-\infty$	$[-\infty$	$[-\infty$	$[-\infty$	$[-\infty$	(NaN
$(-\infty$	$[-\infty$	$(-\infty$	$(-\infty$	$(-\infty$	$[\infty$
$[x$	$[-\infty$	$(-\infty$	$[x + y$	$(x + y$	$[\infty$
$(x$	$[-\infty$	$(-\infty$	$(x + y$	$(x + y$	$[\infty$
$[\infty$	$(NaN$	$[\infty$	$[\infty$	$[\infty$	$[\infty$

$\oplus_{upp}$	$-\infty]$	$y)$	$y]$	$\infty)$	$\infty]$
$-\infty]$	$-\infty]$	$-\infty]$	$-\infty]$	$-\infty]$	NaN)
$x)$	$-\infty]$	$x + y)$	$x + y)$	$\infty)$	$\infty]$
$x]$	$-\infty]$	$x + y)$	$x + y]$	$\infty)$	$\infty]$
$\infty)$	$-\infty]$	$\infty)$	$\infty)$	$\infty)$	$\infty]$
$\infty]$	$NaN)$	$\infty]$	$\infty]$	$\infty]$	$\infty]$

There are corresponding tables for subtraction, multiplication and division which show that the set of ubounds is closed under the four basic arithmetic operations.

Fused Operations and the Mathematica Math Library

Gustafson (2015) provides a basic math library for UNUMs written in Mathematica. It contains built-in functions to compute powers or a more general expression x^y which is an algebraic function for any exactly representable x and y. Also there is an exact dot product which has to be used in case associativity of addition is needed. Similarly, an exact product guarantees the associativity of multiplication. One has to be careful to avoid big ubounds. For example, do not compute x^2 by the product $x \otimes x$ where $\otimes$ denotes multiplication of UNUMs: if x is given by the ubound $x = \{x_{min}, x_{max}\}$ naïve $x \otimes x$ produces $\{\min(x_{min}^2, x_{min}x_{max}, x_{max}^2)$ instead, $\max(x_{min}^2, x_{min}x_{max}, x_{max}^2)\}$ of the much smaller ubound $\{0, \max(x_{min}^2, x_{min}x_{max}, x_{max}^2)\}$

So-called *fused operations* like the exact dot product fight the explosion of the size of intervals notorious for traditional interval arithmetic (Alefeld & Meyer, 2000; Revol, 2015).

Gustafson's math library contains all elementary mathematical functions like powers, exponential, trigonometric, and hyperbolic and their inverses.

Efficient Operations on UNUMs in the Processor

UNUMs are loaded into the processors' usual 64 bit floating point registers in something like an unpacked format in order to speed up operations on UNUMs (see Figure 1). Part of the IEEE 754 flags which were hidden in the processor status word are now directly accessible in the floating point registers.

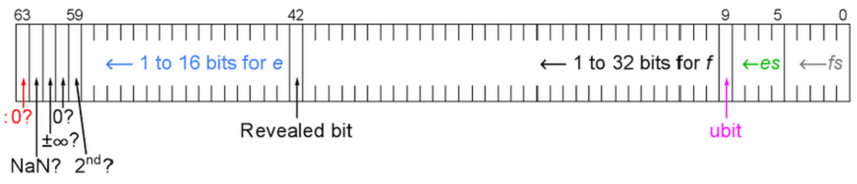

Figure 1. Format and flags of UNUMs in 64 bit processor registers.

So UNUMs are stored efficiently in memory and transferred efficiently to the processor where they are unpacked and operated upon efficiently. Any results that have to be packed are then transferred and stored in the memory efficiently.

Differences between UNUM Arithmetic and IEEE 754 Arithmetic

Let us illustrate the differences between arithmetic on UNUMs and arithmetic on IEEE 754 floating point numbers with the examples of IEEE 754 abnormalities above. $\oplus$, $\ominus$, $\otimes$ and $\oslash$ stand for addition, subtraction, multiplication and division on UNUMs, respectively.

- The term $\frac{99}{7} - \frac{97}{7} - \frac{2}{7}$ is computed by $(99 \oslash 7) \ominus (97 \oslash 7) \ominus (2 \oslash 7)$ and returns the correct interval $(-0.000072479248046875, 0.00017547607421875)$ e.g. in the $\{2,4\}$ environment.
- The two expressions $(10^{16} - 10^{16}) + 1$ and $10^{16} + (-10^{16} + 1)$ are evaluated identically to 1 in the $\{3,6\}$ environment. However, in the $\{3,5\}$ environment,

the intervals computed are too big to be meaningful. But, the environment can automatically be adapted if the need is detected by suitable library functions.

- Computing $\sqrt[2^n]{2}$ iteratively with Unums is done in Mathematica by

  $r = x2u[2];$.
  $Do[r = sqrtu[r]; Print[\text{“}r = \text{''}, view[r]],\ \{i, 1,8\}]$

  E.g. in the {3,3} environment the result is stable after seven iterations returning intervals$(1, u)$ with monotonically decreasing upper bound u. So the results are always greater than 1.
- Solving $\begin{matrix} 0.25510582\,x + 0.52746197\,y = 0.79981812 \\ 0.80143857\,x + 1.65707065\,y = 2.51270273 \end{matrix}$ using Cramer's rule with Unums in the {3,6} environment gives the correct solution (*x, y*)= (–1,2). The same is true in the {3,5} environment if one uses the fused dot product fdotu [{*a*, – *c*}, {*d*, *b*}] to compute the determinant $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$ instead of the naïve expression $(a \otimes d) \ominus (b \otimes c)$.
- Evaluating $H(x) := E\big(Q^2(x)\big)$ with $E(0) := 1$ and $E(x) := \frac{e^x - 1}{x}$ and $Q(x) := \left|x - \sqrt{x^2+1}\right| - \frac{1}{|x+\sqrt{x^2+1}|}$ for *x* = (15,16,17,9999) with
- UNUMs in even the poorest {0,0}environment (one exponent bit and one fraction bit) returns the correct solution (1,1,1,1).
- The Kahan sequence does not go astray but in the example the {3,6} environment nicely converges to the fix point 6.

These few examples should corroborate the claim that UNUMs avoid numerical anomalies. They, at the same time show that careful examination by hand or by hardware is needed to choose the environment in which the required accuracy is achieved.

SCIENTIFIC COMPUTING WITH UNUMS

There are already quite a lot of examples which demonstrate the suitability of UNUMs in scientific computing. They also show that algorithms with UNUMs are different from those with IEEE 754 floating numbers. Algorithms with UNUMs often are declarative and use methods similar to inclusion/exclusion, region growing, lumping, grid refinement, etc. (Gustafson, 2014, 2015a), to solve problems like.

- finite integrals (quadrature determines an area),
- linear equations with exact and inexact coefficients,
- evaluation of polynomials; computation of zeroes, extreme points and fix points,
- computation of the location ϑ of a physical pendulum where speed $\upsilon = \upsilon(\vartheta)$, acceleration a = a($\vartheta$) , and time $t = t(\vartheta)$ are modelled depending on ϑ,

- two- and many-body problems (Gustafson, 2015),
- inverse kinematics (Gustafson, 2016 radical),
- mass-spring-systems, trusses, FFT, CFD, etc.

This list of classical problems in scientific computing alone is in our opinion proof of concept.

THE UNUM MATH LIBRARY

Gustafson provides a library of all elementary mathematical functions for UNUMs written in Mathematica (Gustafson, 2015). It relies on the Mathematica implementations of these elementary mathematical functions. As an unusual departure, the trigonometric functions in this library take their arguments in degrees in order to provide more exact values (Jahnel, 2006, 2013), even in more or less all scientific computing applications these functions take their arguments in radians.

Easy and comfortable deployment of UNUMs in scientific computing relies on the availability of a UNUM math library as is available for IEEE 754 floating point numbers. For example, a library like *Basic Linear Algebra Subprograms, BLAS,* (http://www.netlib.org/blas/) provides highly optimized versions of all sorts of algorithms of linear algebra. The library *Linear Algebra Package, LAPACK,* (http://www.netlib.org/lapack/) can be seen as an extension to BLAS providing, e.g. routines for solving systems of linear equations and linear least squares, eigenvalue problems and singular value decomposition.

Libraries allow easy and comfortable adoption of scientific computing algorithms. Also, many hardware manufacturers offer libraries tailored for their special architectures, for example Intel's *Math Kernel Library, MKL* (https://software.intel.com/en-us/mkl.)

So, in order to support spreading the use of UNUMs a UNUM math library is needed as the examples in the previous section show some algorithms are already available. In order to add more algorithms we focus here quite arbitrarily on root finding algorithms. The problem is to identify zeroes of a given function $y = (x)$ or $w = f(z)$ in a given interval of the real line or in a given rectangular region of the complex plane. Classical algorithms comprise, for example, bisection, secant method (regula falsi), Newton's method, inverse interpolation, Brent's method which combines bisection, the secant method and the inverse quadratic interpolation. All these methods have deficiencies, they do not find complex zeroes or multiple roots. An alternative is based on the idea to use Cauchy's residue calculus (Delves, Lyness, 1966). For any analytic function $f\colon \mathbb{C} \to \mathbb{C}$ and any connected region $R \subset \mathbb{C}$ we have

$$s_n = \frac{1}{2\pi i} \int_C z^n \frac{f'(z)}{f(z)}\, dz = \sum_{i=1}^{N} z_i^n$$

where $z_1, z_2, ..., z_N$ are the zeros of in , according to multiplicity, $C = \partial R$ is the boundary of R and $n \in \mathbb{N}_o$. Now, this formula can be used to approximate every (complex) zero of any analytic function in some regions R including their multiplicity. The idea is to use a quad tree approach. A rectangular region R is quartered and each quarter is examined for the number s_o of zeroes in the quarter. Quarters with no zeroes are discarded; those with zeroes are further quartered. Once there is only one zero in some quarter it is approximated by s_1 and if several zeroes are located in a sufficiently small quarter one uses $s_2, s_3,\dots$ in order to distinguish multiple zeroes from several simpler ones. In case a zero lies on C, lumping the current sub-region with neighboring sub-regions can circumvent this case and allows the approximation of zeros to continue. If one is interested in real zeros only, one just chooses some small banded region around the abscissa.

Let us consider some examples to see how zero finding is done by computing line or contour integrals.

1. The polynomial $f(z) = z^2 - 3z + 2$ has the two real zeros $z_1 = 1$ and $z_2 = 2$. Then $s_o = \frac{1}{2\pi i} \int_C \frac{f'(z)}{f(z)}\, dz = \frac{1}{2\pi i} \int_C \frac{2z-3}{(z-1)(z-2)}\, dz$ so that Cauchy's theorem implies $s_o = g_1(1) = 1$ for $g_1(z) = \frac{2z-3}{z-2.}$ and $C = \partial R$ for any open region R with $1 \in R$ and $2 \notin R \cup C$. A priori knowing helps in the quadrature. We then approximate z_1 by s_1 which by Cauchy's theorem evaluates to $s_1 = \frac{1}{2\pi i} \int_C \frac{z(2z-3)}{(z-1)(z-2)}\, dz = g_2(1) = 1$ for $g_2(z) = \frac{z(2z-3)}{z-2}$. Here, quadrature depends on the requirements of the accuracy of the zeros.
2. The polynomial $f(z) = (z–2)^2$ with the double zero in 2 and $f'(z) = 2z - 4$ shows how to verify or falsify the hypothesis of a multiple zero; if z_o is thought to be the only zero of in some region $2 \in R$ its multiplicity is $m = 2$ if $s_N = \frac{1}{2\pi i} \int_C \frac{z^N 2(z-2)}{(z-2)^2}\, dz = \frac{1}{2\pi i} \int_C \frac{z^N 2}{z-2}\, dz = g_N(2) = 2^N + 2^N$ for $g_N(z) = 2z^N$ holds approximately. Hence, the more of the s_n for $n \geq 1$ approximate the more the hypothesis of f having a zero of multiplicity m in z_o is corroborated.
3. The rational function $f(z) = 1 - \frac{1}{z}$ has the single, simple real zero $z_o = 1$ and $s_o = \frac{1}{2\pi i} \int_C \frac{1/z^2}{(z-1)/z}\, dz = \frac{1}{2\pi i} \int_C \frac{g_o(z)}{z-1}\, dz = g_o(1) = 1$ for $g_o(z) = \frac{1}{z}$, $C = \partial R$

and any connected open region *R* with $1 \in R$ and $0 \notin R \cup C$ and as well as$s_1 = \frac{1}{2\pi i}\int_C \frac{z/z^2}{(z-1)/z}\, dz = \frac{1}{2\pi i}\int_C \frac{g_1(z)}{z-1}\, dz =$ for $g(1) = 1$ for.$g_1(z) = 1$.

4. We want to approximate the greatest real zero of $f(z) = \sin(2z) - z$ with $f'(z) = 2\cos(z) - 1$ which because of $f\left(\frac{\pi}{4}\right) = 1 - \frac{\pi}{4} \approx 0.2146 > 0$ and $f\left(\frac{\pi}{2}\right) = 0 - \frac{\pi}{2} \approx -1.57 < 0$ lies in $(\frac{\pi}{4}, \frac{\pi}{2}) \subset \mathbb{R}$. Figure 2 visualizes the graph of and its real zeroes.

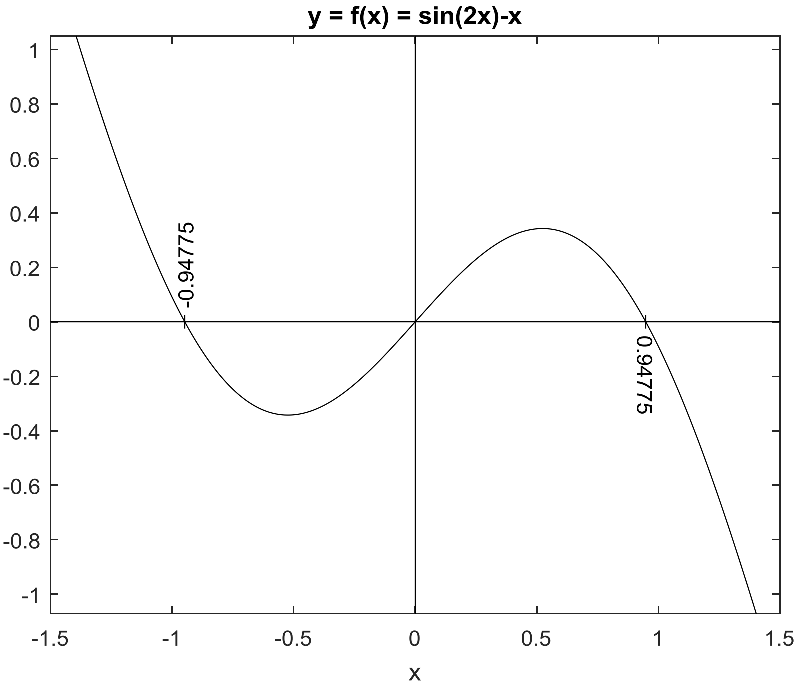

Figure 2. Graph of the real valued function $y = f(x) = \sin(2x) - x$

Then $s_o = 1$ for integration along the edge $C = \partial R$ of the rectangle *R* with lower left vertex $\frac{\pi}{4} - ic$ and upper right vertex $\frac{\pi}{2} + ic$ for any small constant $0 < c \ll 1$. The zero sought-after of is then approximated by $s_1 = \frac{1}{2\pi i}\int_C z\frac{f'(z)}{f(z)}\, dz \approx 0.94775$.

5. Lambert's W-function presents a more serious example. $W(z)$ is defined to be the inverse of $y = f(x) = xe^x$ (shown by the dotted curve). Geometrically obvious is that $W(x)$ has two real branches W_{-1} (shown by the continuous curve) and W_o (shown by the dashed curve). Figure 3 visualizes the graphs of both $f(x)$ as well as of $W(x)$with its two branches W_o and W_{-1}.

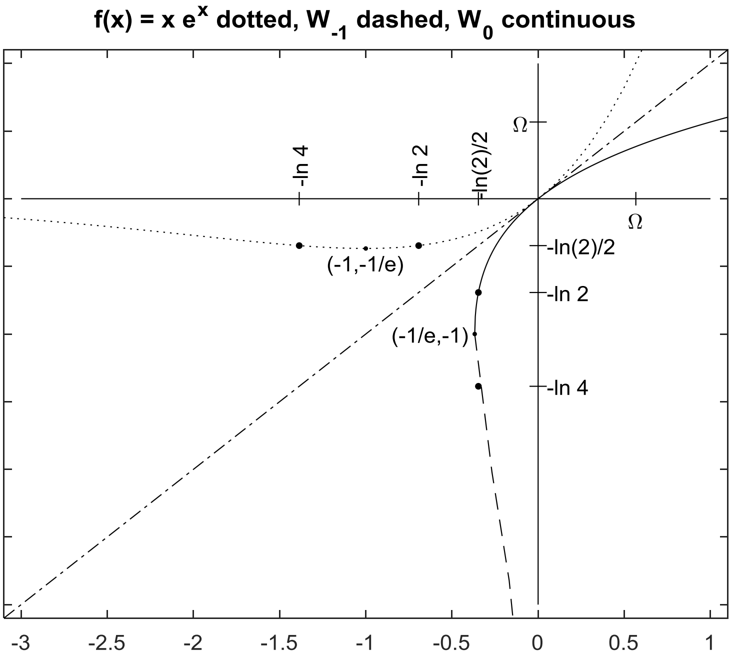

Figure 3. The two branches of the Lambert W-function, inverse of $f(x) = xe^x$.

First, let us approximate $\Omega = W(1)$. Ω is the zero of the function $h(z) = ze^z$. Then our algorithm produces $s_o = 1$ and $\Omega = s_1 \approx 0.5671$ when integrating along the edge $C=\partial R$ of the rectangle R with the lower left vertex $0.5-ic$ and the upper right vertex $0.5 + ic$ for any constant $0 < c \ll 1$. $W(x)$ is double-valued for $x \in (-\frac{1}{e}, 0) \subset \mathbb{R}$, a fact that is e.g. for revealed by $s_o = 2$ when integrating the function $h(z) = ze^z - x_o$ along the edge of a sufficiently large rectangle with, e.g. the lower left vertex $-4.8 - i$ and the upper right vertex $-0.6 + i$ and by $s_o = 1$ when integrating along the edge $C = \partial R$ of, for example, the sub-rectangle R with lower left vertex $-0.8 - i$ and the upper right vertex $-0.6 + i$ to return $s_1 \approx -0.6931$ approximating the exact value $W(x_o) = -\ln(2)$.

These examples demonstrate the feasibility of our algorithm to find zeroes of analytical functions. However, it is still to be decided by which algorithm to approximate the value of the line integrals $s_n = \frac{1}{2\pi i}\int_C z^n \frac{f'(z)}{f(z)} dz = \int_{left} h_n(z)\, dz + \int_{upper} h_n(z)\, dz + \int_{right} h_n(z)\, dz + \int_{lower} h_n(z)\, dz$

where $h_n(z) = z^n \frac{f'(z)}{f(z)}$ and path $C = \partial R = left \cup upper \cup right \cup lower$ consists of the four straight line segments bordering the rectangle R and each of the four line integrals is computed as $\int_{line} h_n(z)\, dz = \int_{t_{min}}^{t_{max}} h_n(\gamma(t))\,\dot{\gamma}(t)\, dt =: \int_{t_{min}}^{t_{max}} (\Re g_n(t) + i\Im g_n(t))\, dt$ for some complex valued analytic function with some analytic parametrization for $\gamma(t)$ of the line segment *line*. There are quadrature rules based on different interpolating functions $g_n(t) := h_n(\gamma(t))\,\dot{\gamma}(t)$ with some analytic parametrization γ(t) for $t \in$ of $[t_{min}, t_{max}]$ the line segment line. There are quadrature rules based on different interpolating functions (e.g. rectangle or midpoint, trapezoidal, Simpson, Boole), and adaptive rules (e.g. Gauss-Legendre, Gauss-Kronrod, Gauss-Lobatto). Depending on the chosen quadrature algorithm, the four complex line integrals can be directly approximated or have to be split into an integral for the real part and one of the imaginary part (which in case of we a priori know to vanish and can therefore disregard).

For a start, in the examples we used the simplest algorithms rectangle rule, the trapezoidal rule the integrated real and the imaginary parts separately. Then, the error of, for example, the midpoint rectangle rule in the interval is bounded by the area $\overline{PQ} \cdot \Delta$ of the quadrangle $\square(ABCD)$ where so to say is inherited by the midpoint rule and Q is inherited by the trapezoidal rule (Needham, 1997).Figure 4 shows the section of the graph of $\Re g_n$ or $\Im g_n$, respectively restricted to the interval $[t_o, t_o + \Delta]$ together with the areas measured by the midpoint and the trapezoidal rule.

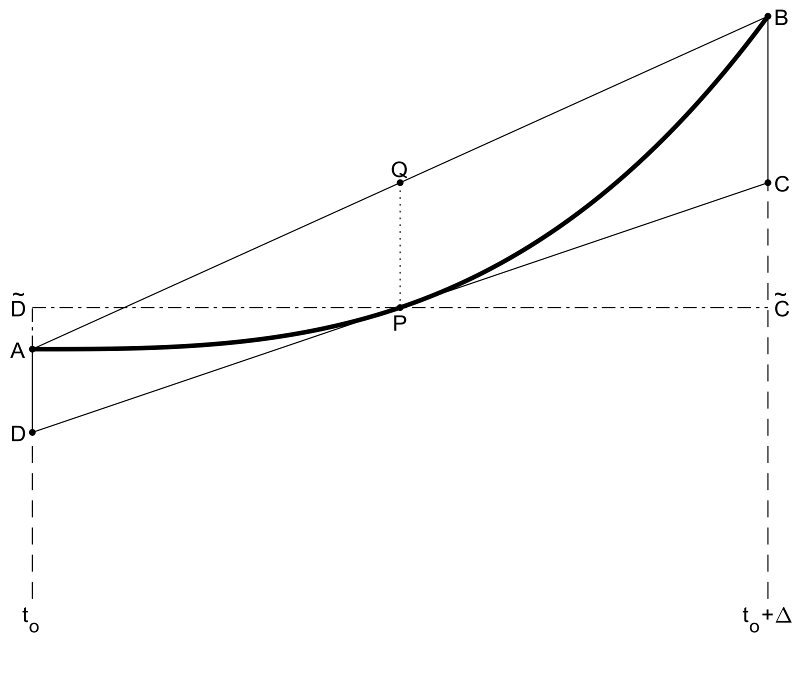

Figure 4. Error bounds for $\int_{t_o}^{t_o+\Delta} \Re g_n(t)\, dt$ and $\int_{t_o}^{t_o+\Delta} \Im g_n(t)\, dt$, respectively.

To guarantee that the graph of the integrand runs in some quadrangle we increase the height of $\square(ABCD)$ by $h(t_o, t_o+\Delta) = \frac{\left(r_{min} - \sqrt{r_{min}^2 - \frac{\ell^2}{4}}\right)}{\cos\alpha} = \frac{\ell}{\Delta}\left(r_{min} - \sqrt{r_{min}^2 - \frac{\ell^2}{4}}\right)$ where $\kappa_{max} = \frac{1}{r_{min}}$ is the maximum curvature of the integrand, $\ell = \sqrt{\Delta^2 + \left(f(t_o+\Delta) - f(t_o)\right)^2}$ is the length and where is the slope angle of the secant.

Similar to (Needham, 1997) we can bound the absolute error for $g = \Re g_n$ and respectively and for some partition $t_o, t_1, \ldots, t_N$ of the integration interval with $\Delta_j = t_{j+1} - t_j$ by

$$error \leq \sum_{j=1}^{N} \left(|\overline{P_j Q_j}| + h_j\right)\Delta_j = \sum_{j=1}^{N} \left|\frac{g(t_j) + g(t_{j+1})}{2} - g\left(\frac{t_j + t_{j+1}}{2}\right)\right| \Delta_j + \sum_{j=1}^{N} h_j \Delta_j$$

where $h_j\Delta_j = \ell_j\left(r_{min} - \sqrt{r_{min}^2 - \frac{\ell_j^2}{4}}\right)$ with $\ell_j = \sqrt{\Delta_j^2 + \left(g(t_{j+1}) - g(t_j)\right)^2}$. The absolute value of both sums can be made as small as desired by controlling $\max_j \Delta_j$.

So we do not need and cannot use tricks similar to the one Gustafson used when dealing with a physical pendulum (Gustafson, 2015).

CONCLUSION AND OUTLOOK

We have demonstrated the shortcomings of the IEEE 754 floating point numbers, arithmetic on such numbers is not trustworthy and the usage of memory and bus bandwidth are not efficient. UNUMs however do not show these deficiencies. However, they make new types of numerical algorithms necessary. We illustrated the problem to design a zero finding algorithm and tailored a suitable one to the use of UNUMs.

At the moment there seems to be only Gustafson's UNUM library written in Mathematica. Of course, libraries which can be used in more common languages are badly needed. But all the same, there are several attempts just to provide such libraries, e.g. in Julia (Gustafson, 2016). Also, there are promising projects under way to implement UNUMs on hardware, e.g. on FPGAs (Gustafson, 2016).

Finally, we do not want to conceal that there are rather heated debates about the virtues of UNUMs between Gustafson (2016a) and Kahan (2016) who by the way was the primary architect of the IEEE 754 standard. At the same time, Gustafson emancipated himself even more from the IEEE 754 heritage

(Gustafson, 2016). Whether his UNUMs 2.0 will be even more profitable than UNUMs 1.0, only applications in scientific computing on high performance computer systems will show.

REFERENCES

Alefeld, G. & Mayer, G., (2000). Interval analysis: Theory and applications. *Journal of Computational and Applied Mathematics*, 121 421–464. Retrieved from http://www-sbras.nsc.ru/interval/Introduction/AleMaSurvey.pdf

Bailey, D. (2012). *Resolving numerical anomalies in scientific computation*. Retrieved from www.davidhbailey.com/dhbpapers/numerical-bugs.pdf

Bailey, D. (2014). *Using high-precision arithmetic to conquer numerical error*. Retrieved from www.davidhbailey.com/dhbtalks/dhb-numerical-error.pdf

Bailey, D. (2015). *Numerical reproducibility in high-performance computing*. Retrieved from www.davidhbailey.com/dhbtalks/dhb-num-repro.pdf

Delves, L. M. & Lyness, J. N. A (1966). *Numerical method for locating the zeros of an analytic function. AMS*. Retrieved from www.ams.org/journals/mcom/1967-21-100/S0025-5718-1967-0228165-4

Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*. Retrieved fromhttps://cr.yp.to/2005-590/goldberg.pdf

Gustafson, J. L. (2014). An energy-efficient and massively parallel approach to valid numerics. *SCAN2014 or ICRAR Seminar*, June 2nd, 2016, Australia. Retrieved from www.johngustafson.net/presentations/UnumArithmetic-ICRARseminar.pdf or www.slideshare.net/insideHPC/unum-computing-an-energy-efficient-and-massively-parallel-approach-to-valid-numerics

Gustafson, J. L. (2015). *The end of error – Unum computing*. CRC Press, Boca Raton, Florida.

Gustafson, J. L. (2015a). The end of numerical error. 22nd *IEEE Symposium on Computer Arithmetic*. June 22nd – 24th 2015 Lyon, France. Retrieved from http://arith22.gforge.inria.fr/slides/06-gustafson.pdf

Gustafson, J. L. (2016). *A radical approach to computation with real numbers*. Retrieved from www.johngustafson.net/pubs/RadicalApproach.pdf

Gustafson, J. L. (2016a). The great debate – Unum arithmetic position statement. *23rd IEEE Symposium on Computer Arithmetic*. Retrieved from http://arith23.gforge.inria.fr/slides/Gustafson.pdf

IEEE 754. (1985). *IEEE standard for binary floating-point arithmetic*. Retrieved from http://ieeexplore.ieee.org/document/30711/

IEEE 754. (2008). *IEEE standard for binary floating-point arithmetic*. Retrieved from http://ieeexplore.ieee.org/document/4610935/

IEEE 1788. (2015). IEEE standard for interval arithmetic. Retrieved from http://ieeexplore.ieee.org/document/7140721/

Jahnel, J. (2006, 2013). *When is the (co)sine of a rational angle equal to a rational number*. Göttingen University. Retrieved from www.uni-math.gwdg.de/jahnel/Preprints/cos.pdf and www.uni-math.gwdg.de/jahnel/Preprints/cos2.pdf

Kahan, W. (2016). A critique of John L. Gustafson's The End of Error – Unum computation and his radical approach to computation with real numbers. *23rd IEEE Symposium on Computer Arithmetic*. Retrieved from http://arith23.gforge.inria.fr/slides/Kahan.pdf

Needham, T. (1997). *Visual complex analysis*. Clarendon Press, Oxford. Retrieved from http://people.math.sc.edu/girardi/m7034/book/VisualComplexAnalysis-Needham.pdf

Risse, Th. (2016). It's time for UNUMs – an alternative to IEEE 754 floats and doubles. In Proc. *34th International Scientific Conf. Science in Practice 2016*, Subotica, Serbia, 50–51, Dec 8th – 9th. Retrieved from www.vts.su.ac.rs/data/files/1/uploads/sip2016/proceedings/10_Risse_SIP2016.pdf

Romik, D. (2016). *Complex analysis lecture notes*. University of California at Davis. Retrieved from www.math.ucdavis.edu/~romik/data/uploads/notes/complex-analysis.pdf

Revol, N. (2015). The (near) future IEEE 1788 standard for interval arithmetic. *8th Small Workshop in Interval Methods*, Prague, June 9th – 11th. Retrieved from http://kam.mff.cuni.cz/conferences/swim2015/slides/revol.pdf