Obliczenia Naukowe

Piotr Kocia, 261924

October 19, 2023

Contents

1 Problem 1

The entire problem consists of several questions and 3 main subproblems:

1. Calculate the machine epsilon (macheps, eps), that is the difference between 1.0 and the next representable value of IEEE754 half, single and double precision floats. As a reference we will be comparing them to Julia's builtin `eps` function.
2. Calculate η which is an equivalent of ϵ for 0.0. We will compare this to Julia's `nextfloat`.
3. Calculate maximum representable value of IEEE754 half, single and double floats and compare to Julia's `floatmax` and values from C's type support library `float.h`.

1.1 Solution

The code may be found in the file `ex1.jl`.

1.2 Results

The results are presented in Table 1, Table 2, Table 3 and Table 4.

1.3 Conclusions

Both Julia and C use the same IEEE754 representation of floats and the results turn out nearly identical. Furthermore:

Type	`calc_macheps`	`eps`	`float.h`
Float16	0.000977	0.000977	-
Float32	1.1920929e-7	1.1920929e-7	1.192092896e-7
Float64	2.220446049250313e-16	2.220446049250313e-16	2.2204460492503131e-16

Table 1: Comparison of machine epsilon calculated experimentally, obtained via `eps` and defined in the C header `float.h`.

Type	`calc_eta`	`nextfloat`
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Table 2: Comparison of experimentally calculated η and the result of `nextfloat`.

- $macheps = 2\epsilon$.
- η is approximately equal MIN_{sub}.
- The result of `floatmin` is approximately equal MIN_{nor}.

2 Problem 2

The goal is to experimentally determine the whether the machine epsilon may be calculated by evaluating the expression $3(4/3 - 1) - 1$, also known as Kahan epsilon (keps).

2.1 Solution

The code may be found in the file `ex2.jl`.

2.2 Results

The results are show in Table 5.

The value of the eps is equal to the absolute value of the keps. The instability in keps stems from the fact that 4/3 cannot be exactly represented in binary, i.e. the mantissa is periodic, and the last 3 bits of the mantissa are `101` in half and double,

Type	`floatmin`	MIN_{nor}
Float32	1.1754944e-38	$1.2 \cdot 10^{-38}$
Float64	2.2250738585072014e-308	$2.2 \cdot 10^{-308}$

Table 3: Comparison of `floatmin` and MIN_{nor}.

Type	`calc_max`	`floatmax`	`float.h`
Float16	6.55e4	6.55e4	-
Float32	3.4028235e38	3.4028235e38	3.40282347e+38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623157e+308

Table 4: Comparison of calculated experimentally maximum float value, obtained via `eps` and defined in the C header `float.h`.

and `011` in single, meaning single is rounded up and overestimated, while half and double are underestimated.

Type	keps	`eps`
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Table 5: Comparison of Kahan's epsilon and the function `eps`.

2.3 Conclusions

3 Problem 3

The goal is to experimentally prove that all float numbers in $[1, 2]$ are uniformly distributed with $\delta = 2^{-52}$. Furthermore, prove that in $[\frac{1}{2}, 1]$ and $[2, 4]$ number are also uniformly distributed.

3.1 Solution

The code may be found in `ex3.jl`. The idea is to take the difference between the start of the interval `v` and `nextfloat(v)`.

3.2 Results

The resulting steps of adjacent intervals, as expected, differ in the exponent by exactly 1 (especially visible in the binary representation). The actual values are shown in Table 6.

Range	Step	Binary Representation
$[0.5, 1.0[$	1.1102230246251565e-16	0011110010100000000000...
$[1.0, 2.0[$	2.220446049250313e-16	0011110010110000000000...
$[2.0, 4.0[$	4.440892098500626e-16	0011110011000000000000...

Table 6: Steps in the intervals $[1.0, 2.0[$, $[0.5, 1[$, $[2.0, 4[$.

3.3 Conclusions

The results unanimously and undeniably prove that the numbers are uniformly spaced in their respective intervals. In $[1, 2]$ the step $\delta = 2^{-52}$, in $[\frac{1}{2}, 1]$ it is $\delta = 2^{-53}$ and in $[2, 4]$ $\delta = 2^{-51}$. Furthermore, the results may be generalised to any interval where the lower and upper (exclusive) bounds have the same exponent in the IEEE754 representation. The only changing part will be the mantissa as the exponent is fixed in such an interval, hence the step is equal to a number with the same exponent and mantissa consisting of only a single set bit at the lowest position.

4 Problem 4

Find the least $x > 1$ such that $x \cdot \frac{1}{x} \neq 1$.

4.1 Solution

We iterate from 1 using `nextfloat` to increment the value and test each one for the aforementioned condition. The code may be found in `ex4.jl`.

4.2 Results

The first value found is 1.000000057228997 and indeed

$$1.000000057228997 \cdot \frac{1}{1.000000057228997} = 0.9999999999999999$$

which satisfies the above condition.

4.3 Conclusions

The division operation introduces a small but significant error to our calculations, hence it is invalid to compare two float numbers directly.

5 Problem 5

Calculate the inner product of two vectors

$$\begin{aligned} x =& [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957] \\ y =& [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049] \end{aligned}$$

using 4 different methods: naive, reverse naive, sorted partial sums in descending order, sorted partial sums in ascending order.

5.1 Solution

The code may be found in `ex5.jl`. Generic functions are used to reduce code duplication.

5.2 Results

5.3 Conclusions

The exact result is $-1.00657107000000 \cdot 10^{-11}$ and as may be seen the results in Table 7 differ greatly. This highlights the importance of the order of operations as well as accumulation of error.

	Float32	Float64
Naive Forward	-0.4999443	1.0251881368296672e-10
Naive Backward	-0.4543457	-1.5643308870494366e-10
Partial Descending	-0.5	0.0
Partial Ascending	-0.5	0.0

Table 7: Comparison of the results of the different inner product methods.

6 Problem 6

Calculate and compare the results of two functions

$$\begin{aligned} f(x) =& \sqrt{x^2+1} - 1 \\ g(x) =& x^2 / \left(\sqrt{x^2+1} + 1\right) \end{aligned}$$

for $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$.

6.1 Solution

The code may be found in `ex6.jl`. The functions are iteratively evaluated until $8^{-i} > 0$.

6.2 Results

The results are shown in Table 8.

6.3 Conclusions

Although mathematically identical, the function in the form of g yields more accurate results. The first 8 iterations give very similar results, however, after that point f yields nothing but 0. This is dictated by the simple fact that $\lim_{x\to 0} \sqrt{x^2+1} = 1$ and subtracting very close values is subject to a substantial error as may be seen in Table 8. On the other hand, g does not suffer from this problem and as a result yields much more accurate results.

i	x	$f(x)$	$g(x)$
-1	0.125	0.0077822185373186414	0.0077822185373187065
-2	0.015625	0.00012206286282867573	0.00012206286282875901
-3	0.001953125	1.9073468138230965e-6	1.907346813826566e-6
-4	0.000244140625	2.9802321943606103e-8	2.9802321943606116e-8
-5	3.0517578125e-5	4.656612873077393e-10	4.6566128719931904e-10
-6	3.814697265625e-6	7.275957614183426e-12	7.275957614156956e-12
-7	4.76837158203125e-7	1.1368683772161603e-13	1.1368683772160957e-13
-8	5.960464477539063e-8	1.7763568394002505e-15	1.7763568394002489e-15
-9	7.450580596923828e-9	0.0	2.7755575615628914e-17
...	...	...	...
-172	4.661462957000129e-156	0.0	1.086461844974e-311
-173	5.826828696250162e-157	0.0	1.69759663277e-313
-174	7.283535870312702e-158	0.0	2.65249474e-315
-175	9.104419837890877e-159	0.0	4.144523e-317
-176	1.1380524797363597e-159	0.0	6.4758e-319
-177	1.4225655996704496e-160	0.0	1.012e-320
-178	1.778206999588062e-161	0.0	1.6e-322
-179	2.2227587494850775e-162	0.0	0.0

Table 8: Comparison of the results of $f(x)$ and $g(x)$.

7 Problem 7

Use the formula

$$f'(x) \approx \widetilde{f}'(x) = \frac{f(x+h) - f(x)}{h}$$

to approximate the value of $f(x) = \sin x + \cos 3x$ and compare to the exact value. Additionally, vary the h parameter to observe how the result changes.

7.1 Solution

The code may be found in `ex7.jl`.

7.2 Results

The exact value of $f'(1) = 0.11694228168853815$. The values of $\widetilde{f}'$ with varying h is in Table 9.

n	$\widetilde{f}'(1)$	$\lvert f'(1) - \widetilde{f}'(1)\rvert$	$1+h$
0	2.0179892252685967	1.9010469435800585	2.0
1	1.8704413979316472	1.753499116243109	1.5
2	1.1077870952342974	0.9908448135457593	1.25
3	0.6232412792975817	0.5062989976090435	1.125
4	0.3704000662035192	0.253457784514981	1.0625
5	0.24344307439754687	0.1265007927090087	1.03125
6	0.18009756330732785	0.0631552816187897	1.015625
...	...	...	...
46	0.109375	0.007567281688538152	1.0000000000000142
47	0.109375	0.007567281688538152	1.000000000000007
48	0.09375	0.023192281688538152	1.0000000000000036
49	0.125	0.008057718311461848	1.0000000000000018
50	0.0	0.11694228168853815	1.0000000000000009
51	0.0	0.11694228168853815	1.0000000000000004
52	-0.5	0.6169422816885382	1.0000000000000002
53	0.0	0.11694228168853815	1.0
54	0.0	0.11694228168853815	1.0

Table 9: Changing value of $\widetilde{f}'$ with respect to h.

7.3 Conclusions