Scientific Calculations - Exploration of Arithmetic

Mateusz Pełechaty

23 October 2022

1 Exercise - Basic Exploration

1.1 Machine Epsilon

Machine epsilon (*macheps*) is the smallest number x such that $1+x > 1$ and $rd(1+x) = 1+x$.
Find machine epsilon with Julia and compare results with the function eps and with values from float.h

Solution can be found in

```
./zad1/find_macheps.jl
./zad1/epsilons.c
```

Method and results

It was calculated by setting $macheps := 1$ and then if $1 + macheps > 1$, then $macheps$ is divided by 2.

Exercise 1.1	Macheps	Eps	Float.h
Float16	0.000977	0.000977	
Float32	1.1920e-7	1.1920e-7	1.1920e-07
Float64	2.2204e-16	2.2204e-16	2.2204e-16

Conclusion

Calculating macheps by me, *eps(type)* function and Float.h constants provide the same values

1.2 Eta

Eta (η) is the smallest number such that $\eta > 0$.
Find η and compare it with nextfloat(0.0) and MIN_{sub}
Tests should be made for **Float16**, **Float32**, **Float64**

Solution can be found in

```
./zad1/find_eta.jl
```

Method and results

It was calculated by setting $\eta := 1$ and then dividing by 2 until $\frac{\eta}{2} > 0$ MIN_{sub} values are taken from *W. Kahan's* book

Exercise 1.2	η	nextfloat	MIN_{sub}
Float16	6.0e-8	6.0e-8	
Float32	1.0e-45	1.0e-45	1.3e-45
Float64	5.0e-324	5.0e-324	4.9e-324

Conclusion

nextfloat(0.0) and my method of calculating η provide the same values.
Values are almost the same as MIN_{sub}

1.3 Questions

Q: What is difference between *macheps* and arithmetic precision (ϵ)?
A: Macheps is the smallest number that meets condition: $1 + macheps > 1$ and $rd(1+macheps) = 1+macheps$. We can also say that $macheps = \beta^{1-t}$. ϵ on the other hand is biggest relative error that can happen due to rounding in arithmetic. So it is the smallest number ϵ, that meets condition $\epsilon \geq \delta = \frac{|rd(x)-x|}{x}$ for some number x. It was calculated in the lecture that

$\epsilon = \frac{1}{2}\beta^{1-t}$
It follows from here that $\epsilon = \frac{macheps}{2}$
Q: What is difference between η and (MIN_{sub})?
A: MIN_{sub} is minimal subnormal number. η is defined by minimal number bigger than 0. They should be the same, but there are little differences between them
Q: What is the returned by *floatmin* and what is it's connection with MIN_{nor}
A: They are the same value as seen in table below.
Values of MIN_{nor} are taken from *W. Kahan's* book
Values of floatmin are calculated in ./zad1/floatmin.jl

Q3	floatmin	MIN_{nor}
Float16	6.104e-5	
Float32	1.1755e-38	1.2E-38
Float64	2.2251e-308	2.2E-308

1.4 FloatMax

Calculate maximum possible number for Float16, Float32, Float64. Compare values with the ones returned by function *floatmax* and with data

Solution can be found in

```
./zad1/floatmax.jl
```

Method and results

It was calculated by $max1 := 4$, $max2 := 2$ and $max3 := 1$. Variables are doubled until $max1 == max2$. It means that they are infinity. Then I am returning $max3$

Exercise 1.4	my max	floatmax	W. Kahan's MAX
Float16	3.277e4	6.55e4	
Float32	1.701e38	3.403e38	3.4 E38
Float64	8.988e307	1.798e308	1.8 E308

Conclusion

We can see that *floatmax* is the same as *W. Kahan's* Max
My method of calculating maximum gets me wrong because doubling number reaches infinity. So $mymax = \frac{1}{2} \cdot floatmax$

2 Exercise - Tricky Macheps Calculation

Check in *Julia* if $3 \cdot (4/3 - 1) - 1$ is Machine Epsilon. Conduct experiments for Float16, Float32 and Float64

Solution can be found in

```
./zad2/calculate_macheps.jl
```

Method and results

Macheps was calculated by *nextfloat*(1.0) and W Kahan's Macheps is calculated by specified above.

Exercise 2	Macheps	Kahan Macheps
Float16	0.000977	-0.000977
Float32	1.1920e-7	1.1920e-7
Float64	2.2204e-16	-2.2204e-16

Conclusion

We can see that $macheps = 3 \cdot (4/3 - 1) - 1$ with an accuracy of up to sign

3 Exercise - Number Distribution

Check experimentally that every number $x \in [1, 2)$ is distributed evenly with step $\delta = 2^{-52}$ It also means it can be represented by $x = 1 + k\delta$ for $\delta = 2^{-52}$ and $k = 1, 2, ..., 2^{52} - 1$

Solution can be found in

```
./zad3/experiment.jl
```

Method and results

Experiment description:

1. Pick random number $x \in [1, 2)$.
2. Make $next := nextfloat(x)$
3. Print their bitstrings

Example results below:

```
num      : 1.3881779261232619
next(num): 1.388177926123262
num:       0011111111110110001101011111101000001110100110111010011111111110
next(num): 0011111111110110001101011111101000001110100110111010011111111111
```

Conclusion

We can see that mantissa of next(num) is β^{1-t} bigger than mantissa of num and exponent stays the same. Here $t = 53$ and $c = 0$ so difference is $\beta^{1-t} = \beta^{-52}$
We can say that $num = 2^c \cdot M$. Then $next(num) = 2^c \cdot (M + \beta^{1-t})$
Then difference is $next(num) - num = 2^c \cdot B^{1-t}$
Q: What is the distribution of floats in range $[\frac{1}{2}, 1)$? How can they be represented?
A: Distribution is with step $\delta = \beta^{-53}$ and they can be represented as $x = 1 + k\delta$ where $k = 1, 2, ..., 2^{52} - 1$
Q: What is it in range $[2, 4)$? How can they be represented?
A: Distribution is with step $\delta = \beta^{-51}$ and they can be represented as $x = 1 + k\delta$ where $k = 1, 2, ..., 2^{52} - 1$

4 Exercise Reciprocal of a number

Find smallest number x such that $x \cdot 1/x \neq 1$

Solution can be found in

```
./zad4/taskB.jl
```

Method and results

We start with $x := 1$ and we do $x := nextfloat(x)$ until $x \cdot 1/x = 1$

```
Found: 1.000000057228997,
bitstring: 0011111111110000000000000000000000001111010111001011111100101010
```

5 Exercise - Scalar Product

Calculate scalar product of x and y in float32 and float64
$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$
$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$
Add products of x_i and y_i in 4 ways.

1. $\sum_{i=0}^{4} x_i \cdot y_i$
2. $\sum_{i=0}^{4} x_{4-i} \cdot y_{4-i}$
3. Add sorted by absolute value of $x_i \cdot y_i$ decreasing
4. Add sorted by absolute value of $x_i \cdot y_i$ ascending

Solution can be found in

```
./zad5/solution.jl
```

Results

Real value: 1.0066e-11

	Float32	Float64
Front	-0.4999443	1.0252e-10
Back	-0.4543457	-1.5643e-10
BigToSmall	-0.5	0.0
SmallToBig	-0.5	0.0

Conclusion

Summation order matters and can change result by a lot

6 Exercise - Equivalent functions

Calculate in **Julia** in **Float64** following functions

$$f(x) = \sqrt{x^2+1} - 1$$

$$g(x) = x^2/(\sqrt{x^2+1} + 1)$$

for $x = 8^{-k}$ for $k = 1, 2, 3, ...$
Why is computer giving different results even though $f == g$?
Which one is giving better results?

Solution can be found in

```
./zad6/tests.jl
```

x	$f(x)$	$g(x)$
8^{-1}	0.0077822185373186414	0.0077822185373187065
8^{-2}	0.00012206286282867573	0.00012206286282875901
8^{-3}	1.9073468138230965e-6	1.907346813826566e-6
8^{-4}	2.9802321943606103e-8	2.9802321943606116e-8
8^{-5}	4.656612873077393e-10	4.6566128719931904e-10
8^{-6}	7.275957614183426e-12	7.275957614156956e-12
8^{-7}	1.1368683772161603e-13	1.1368683772160957e-13
8^{-8}	1.7763568394002505e-15	1.7763568394002489e-15
8^{-9}	0.0	2.7755575615628914e-17
8^{-10}	0.0	4.336808689942018e-19
8^{-11}	0.0	6.776263578034403e-21
8^{-12}	0.0	1.0587911840678754e-22

Conclusion

$f(x)$ and $g(x)$ are giving different results due to rounding errors.
$\sqrt{x^2+1}$ cannot be calculated as accurate as we would like because step in range $[1, 2)$ is $\delta = 2^{-52}$. Because of it, we cannot rely on this value. As accuracy in range $[0, 1)$ is much bigger, $g(x)$ conserves it's accuracy for longer by making division with x^2

7 Exercise - Derivative

Compare approximations errors of function $f(x) = sin(x) + cos(3x)$ for approximation $f'(x_0) \approx \overline{f'}(x_0) = \frac{f(x_0+h)-f(x_0)}{h}$ in point $x_0 = 1$ for $h = 2^{-n}$, for $n = 1, 2, 3..., 54$

Why at some point decreasing h doesn't improve approximation of derivative?

How does behave $1 + h$?

Solution can be found in

```
./zad7/calculations.jl
```

$f'(x) = 0.11694228168853815$

n	approx derivative	error	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
...	...	...	...
...	...	...	...
24	0.11694252118468285	2.394961446938737e-7	1.0000000596046448
25	0.116942398250103	1.1656156484463054e-7	1.0000000298023224
26	0.11694233864545822	5.6956920069239914e-8	1.0000000149011612
27	0.11694231629371643	3.460517827846843e-8	1.0000000074505806
28	0.11694228649139404	4.802855890773117e-9	1.0000000037252903
29	0.11694222688674927	5.480178888461751e-8	1.0000000018626451
30	0.11694216728210449	1.1440643366000813e-7	1.0000000009313226
31	0.11694216728210449	1.1440643366000813e-7	1.0000000004656613
...	...	...	...
...	...	...	...
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

Conclusion

We see that best accuracy is achieved for $n = 28$ Accuracy of approximation decreases after that because as we are decreasing h, there is bigger rounding error for $1 + h$. It makes $f(1 + h)$ calculate with error