Obliczenia naukowe
Sprawozdanie 1

Józef Piechaczek

19 października 2019

1 Zadanie 1

1.1 Program 1

Program 1 polega na wyznaczeniu iteracyjnie epsilonów maszynowych dla typów **Float16**, **Float32**, **Float64** zgodnych ze standardem **IEEE 754** i porównaniu ich z wartościami zwracanymi przez funkcje **eps(Float16)**, **eps(Float32)**, **eps(Float64)** oraz z danymi zawartymi w pliku nagłówkowym float.h dowolnej instalacji języka C. Wyniki przedstawiam w poniższej tabeli:

	myEps	eps	float.h
Float16	0.000977	0.000977	-
Float32	1.1920929e-7	1.1920929e-7	1.192093e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.220446e-16

Można zatem zauważyć, że obliczone iteracyjnie wartości, wartości uzyskane za pomocą funkcji w języku Julia oraz wartości z pliku float.h są równe.

Aby sprawdzić związek liczby eps z precyzją arytmetyki obliczyłem wartość ϵ dla podanych typów zgodnie ze wzorem:

$$\epsilon = 0.5 * \beta^{1-t}$$

Wyniki przedstawiam w poniższej tabeli:

	t	ϵ
Float16	11	0.00048828125
Float32	24	5.9604644775390625e-8
Float64	53	1.1102230246251565404236316680908203125e-16

Można zatem zauważyć, iż epsilon maszynowy nie odpowiada precyzji arytmetyki. Wynika to z faktu iż w **IEEE 754** domyślnie zakładamy wartość jeden z przodu (dla liczb znormalizowanych).

1.2 Program 2

Program 2 polega na wyznaczeniu iteracyjnie liczby *eta* takiej, że $eta > 0.0$ dla podanych typów i porównaniu ich z wartościami zwracanymi przez funk-

cje **nextfloat(Float16(0.0))**, **nextfloat(Float32(0.0))**, **nextfloat(Float64(0.0))**. Wyniki przedstawiam w poniższej tabeli:

	myEta	eta
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Można zatem zauważyć, że obliczone wartości są równe rzeczywistym.

Liczba *eta* jest równa liczbie MIN_{sub}. Wynika to z faktu, iż liczba *eta* będąc kolejną liczbą po 0.0 jest również najmniejszą liczbą zdenormalizowaną. Potwierdza to funkcja **bitstring** - można zauważyć że cecha składa się z samych zer (więc jest zdenormalizowana), a mantysa z jednej jedynki na najmniej znaczącym bicie. Potwierdza to równiez funkcja **issubnormal**.

Funkcje **floatmin(Float32)** i **floatmin(Float64)** zgodnie z dokumentacją zwracają "najmniejszą liczbę znormalizowaną". Potwierdza to eksperyment polegający na zamienieniu liczby o zapisie bitowym w **IEEE 754** składającym się z samych zer i jednej jedynki na najmniej znaczącym bicie w cesze, na odpowiadający jej typ. Uzyskane wartości były równe.

1.3 Program 3

Program 3 polega na wyznaczeniu iteracyjnie liczby MAX w podanych typach i porównaniu ich z wartościami zwracanymi przez funkcje **floatmax(Float16)**, **floatmax(Float32)**, **floatmax(Float64)** oraz z danymi zawartymi w pliku nagłówkowym float.h dowolnej instalacji języka C. Wyniki eksperymentu przedstawia poniższa tabela:

	myMax	max	float.h
Float16	6.55e4	6.55e4	-
Float32	3.4028235e38	3.4028235e38	3.402823e38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.797693e308

Można zatem zauważyć, że obliczone wartości są równe rzeczywistym.

Liczbę MAX wyznaczałem iteracyjnie mnożąc liczbę n o 2 w każdej iteracji, począwszy od **prevfloat(2.0)**. Zacząłem od takiej liczby, gdyż gwarantowała ona mantysę składającą się z samych jedynek, co pozwalało na uzyskanie maksymalnej liczby.

2 Zadanie 2

Celem zadania 2 jest sprawdzenie, czy epsilon maszynowy da się otrzymać obliczając wyrażenie $3(4/3 - 1) - 1$ w arytmetyce zmiennopozycyjnej.

Test wykonałem dla typów **Float16**, **Float32** i **Float64**. Wyniki przedstawiam w poniższej tabeli:

	epsilon (Kahan)	epsilon maszynowy
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Można dostrzec że powyższa metoda pozwala na dokładne określenie wartości epsilona maszynowego, niekiedy z różnicą znaku. Wynika to z faktu, iż liczbę 4/3 nie da się przedstawić w dokładny sposób w podanych arytmetykach. Dzielenie to jest jedynym źródłem błędu w powyższym wyrażeniu - odejmowanie wyrażeń, gdy cecha jest taka sama jest dokładne, podobnie w przypadku mnożenia. Końcowy wynik zatem jest równy początkowemu błędowi, który jest naszym epsilonem maszynowym.

3 Zadanie 3

Celem zadania 3 jest sprawdzenie eksperymentalnie w języku Julia, że w arytmetyce **Float64** liczby zmiennopozycyjne są równomiernie rozmieszczone w $[1, 2]$ z krokiem $\delta = 2^{-52}$.

Wiemy, iż liczba 1 jest zapisana w systemie **IEEE-754 Float64** jako

0 01111111111 0000000000000000000000000000000000000000000000000000

Oznacza to iż cecha jest równa 0 (po przesunięciu o bias), czyli zgodnie ze standardem liczbę można zapisać jako $2^0 * 1$. Następna liczba w tej arytmetyce powstanie po dodaniu do mantysy wartości 1. Wiedząc, iż na mantysę przeznaczymy 52 bity, możemy zauważyć, że mantysa zwiększy wtedy swoją wartość o $\delta = 2^{-52}$. Cała liczba również zwiększy swoją wartość o podaną δ, gdyż 2^c, przez które przemnażamy wartość mantysy, w tym przykładzie jest równe 1.

W celu sprawdzenie powyższego rozumowania napisałem program w Julii, dodający do wartości 1.0 podaną deltę określoną ilość razy. Za pomocą funkcji bitstring można zauważyć, że do mantysy po każdej iteracji dodawana jest wartość 1, co zgadza się z powyższym rozumowaniem.

Powyższe rozumowanie można przeprowadzić w analogiczny sposób dla przedziałów $[\frac{1}{2}, 1]$ oraz $[2, 4]$. Liczba $\frac{1}{2}$ zapisana jest jako

0 01111111110 0000000000000000000000000000000000000000000000000000

czyli cecha ma wartość -1, a mantysa 1. Gdy zwiększamy mantysę o 1 bit jej wartość rośnie o $\delta = 2^{-52}$, jednak jest ona przemnażana przez $2^c = 2^{-1}$ co sprawia że liczba zwiększa swoją wartość o 2^{-53}.

Analogicznie dla liczby 2 cecha ma wartość 1 co sprawia, że liczba zwiększa swoją wartość o $2^{-52+1} = 2^{-51}$

4 Zadanie 4

Celem zadania 4 jest znalezienie eksperymentalnie w arytmetyce **Float64** zgodnej ze standardem **IEEE-754** takiej liczby x w przedziale $1 < x < 2$, że $x * \frac{1}{x} \neq 1$ tj. $fl(xfl(\frac{1}{x})) \neq 1$.

W tym celu napisałem program, który począwszy od $x = nextfloat(1.0)$ sprawdza spełnianie podanego równania przez kolejne liczby z arytmetyki.

Najmniejszą liczbą z przedziału, która spełnia powyższą nierówność jest 1.000000057228997, a najmniejszą liczbą w arytmetyce jest $-1.7976931348623157e308$

5 Zadanie 5

Zadanie 5 polega na przeprowadzeniu eksperymentu dotyczącego obliczania iloczynu skalarnego dwóch wektorów:

$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$

$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$

za pomocą czterech algorytmów

1. w przód
2. w tył
3. od największego do najmniejszego
4. od najmniejszego do największego

Wyniki eksperymentu przedstawia poniższa tabela:

	Float32	Float64
Algorytm 1	-0.4999443	1.0251881368296672e-10
Algorytm 2	-0.4543457	-1.5643308870494366e-10
Algorytm 3	-0.5	0.0
Algorytm 4	-0.5	0.0

Iloczyn skalarny w rzeczywistości wynosi $-1.00657107000000 * 10^{-11}$. Policzmy błąd względny dla otrzymanych wyników. Poniższa tabela przedstawia zaokrąglone wartości błędów względnych.

	Float32	Float64
Algorytm 1	$4.9668 * 10^{12}$	$1.1185 * 10^{3}$
Algorytm 2	$4.5138 * 10^{12}$	$1.4541 * 10^{3}$
Algorytm 3	$4.9674 * 10^{12}$	$1.0000 * 10^{2}$
Algorytm 4	$4.9674 * 10^{12}$	$1.0000 * 10^{2}$

Otrzymane wyniki okazały się nieintuicyjne. Algorytm "od najmniejszego do największego" wypadł najgorzej, mimo iż powinien być najdokładniejszy.

Algorytm "od największego do najmniejszego" wypadł tak samo, mimo iż wydawał się najgorszym algorytmem.

Różnice w błędach dla typów **Float32** i **Float64** okazały się znaczące - zgodnie z oczekiwaniami wyniki we **Float64** były dużo dokładniejsze.

6 Zadanie 6

Zadanie 6 polega na obliczeniu w Julii w arytmetyce **Float64** wartości następujących funkcji:

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

dla kolejnych wartości elementu $x = 8^{-1}, 8^{-2}, 8^{-3}, ...$

Wyniki przestawiam w poniższej tabeli:

	$f(x)$	$g(x)$
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	$4.336808689942018e-19$

Mimo iż funkcje f i g są sobie równe otrzymujemy inne wyniki. Można zauważyć iż od $x = 8^{-9}$ funkcja f zwraca wartość 0.0, co oznacza błąd względny wynoszący 100%. Wynika to z faktu, iż w pewnym momencie wartość x^2 jest tak niewielka, iż przy wykonywaniu działania $x^2 + 1$ jest ona pochłaniana przez jedynkę. Zarówno w funkcji f jak i g wartość pierwiastka przyjmuje wtedy wartość 1. Ponieważ funkcja f nie zawiera zmiennej x w innym miejscu traci ona całkowicie informacje o wprowadzonej zmiennej i po wykonaniu

odejmowania przyjmuje wartość 0.0. Funkcja g natomiast zawiera zmienną x w innym miejscu - w liczniku znajduje się wyrażenie x^2. Wyrażenie to nie jest pochłaniane przez inna wartość, jak ma to miejsce w przypadku funkcji f i mimo, iż w mianowniku mamy wartość 2.0, pozwala ono na dokładniejsze oszacowanie liczonej wartości .

7 Zadanie 7

Celem zadania 7 jest obliczenie przybliżonej pochodnej funkcji $f(x) = sinx + cos3x$ w punkcie $x_0 = 1$ za pomocą następującego wzoru:

$$f'(x_0) \approx \widetilde{f}'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

oraz oszacowanie błędów $|f'(x_0) - \widetilde{f}'(x_0)|$ dla $h = 2^{-n}, n = 0, 1, 2, ..., 54$

Wyniki przedstawiam w poniższych tabelach:

n	h	1+h
0	1.0	2.0
1	0.5	1.5
2	0.25	1.25
3	0.125	1.125
4	0.0625	1.0625
5	0.03125	1.03125
6	0.015625	1.015625
7	0.0078125	1.0078125
8	0.00390625	1.00390625
9	0.001953125	1.001953125
10	0.0009765625	1.0009765625
11	0.00048828125	1.00048828125
12	0.000244140625	1.000244140625
13	0.0001220703125	1.0001220703125
14	6.103515625e-5	1.00006103515625
15	3.0517578125e-5	1.000030517578125
16	1.52587890625e-5	1.0000152587890625
17	7.62939453125e-6	1.0000076293945312
18	3.814697265625e-6	1.0000038146972656
41	4.547473508864641e-13	1.0000000000004547
42	2.2737367544323206e-13	1.0000000000002274
43	1.1368683772161603e-13	1.0000000000001137
44	5.684341886080802e-14	1.0000000000000568
45	2.842170943040401e-14	1.0000000000000284
46	1.4210854715202004e-14	1.0000000000000142
47	7.105427357601002e-15	1.000000000000007
48	3.552713678800501e-15	1.0000000000000036
49	1.7763568394002505e-15	1.0000000000000018
50	8.881784197001252e-16	1.0000000000000009
51	4.440892098500626e-16	1.0000000000000004
52	2.220446049250313e-16	1.0000000000000002
53	1.1102230246251565e-16	1.0
54	5.551115123125783e-17	1.0

n	$\widetilde{f'}(x_0)$	error
0	2.0179892252685967	1.9010469435800585
1	1.8704413979316472	1.753499116243109
2	1.1077870952342974	0.9908448135457593
3	0.6232412792975817	0.5062989976090435
4	0.3704000662035192	0.253457784514981
5	0.24344307439754687	0.1265007927090087
6	0.18009756330732785	0.0631552816187897
7	0.1484913953710958	0.03154911368255764
8	0.1327091142805159	0.015766832591977753
9	0.1248236929407085	0.007881411252170345
10	0.12088247681106168	0.0039401951225235265
11	0.11891225046883847	0.001969968780300313
12	0.11792723373901026	0.0009849520504721099
13	0.11743474961076572	0.0004924679222275685
14	0.11718851362093119	0.0002462319323930373
15	0.11706539714577957	0.00012311545724141837
16	0.11700383928837255	6.155759983439424e-5
17	0.11697306045971345	3.077877117529937e-5
18	0.11695767106721178	1.5389378673624776e-5
41	0.116943359375	1.0776864618478044e-6
42	0.11669921875	0.0002430629385381522
43	0.1162109375	0.0007313441885381522
44	0.1171875	0.0002452183114618478
45	0.11328125	0.003661031688538152
46	0.109375	0.007567281688538152
47	0.109375	0.007567281688538152
48	0.09375	0.023192281688538152
49	0.125	0.008057718311461848
50	0.0	0.11694228168853815
51	0.0	0.11694228168853815
52	-0.5	0.6169422816885382
53	0.0	0.11694228168853815
54	0.0	0.11694228168853815

Dokładna pochodna w naszym przykładzie wynosi $f'(x_0) = 0.11694228168853815$. W zadaniu 7 można dostrzec, iż zmniejszanie liczby h zwiększa dokładność wyniku aż do pewnego momentu, od którego zmniejszania obniża dokładność. Wynika to z faktu, iż dla coraz mniejszego h znacząco wzrasta względny błąd przedstawienia tej liczby w arytmetyce i w pewnym momencie staje się na tyle duży, iż niweluje korzyści z rozpatrywania pochodnej na coraz mniejszym przedziale. Przykładowo dla $n = 51$ błąd względny przedstawienia h wynosi aż $\delta_{51} = 26.98\%$. Obserwując wartość $1 + h$ można zauważyć, że od pewnego n, $1 + h = 1$. Wartość x_0 pochłania wtedy wartość h w wyrażeniu $f(x_0 + h)$ co powoduje że w liczniku mamy wyrażenie $f(x_0) - f(x_0)$, a więc całe wyrażenie przyjmuje wartość 0.0.