Obliczenia Naukowe

Lista nr 1

Eryk Krupa

244993

Politechnika Wrocławska 2019

Zadanie 1

Wartości wyznaczonych epsilonów maszynowych okazały się w każdym typie identyczne z wartością zwracaną przez funkcję eps().

Epsilon Maszynowy	Float16 (half)	Float32 (single)	Float64 (double)
Wyznaczony	0.000977	1.1920929e-7	2.220446049250313e-16
Julia- eps()	0.000977	1.1920929e-7	2.220446049250313e-16
C- float.h	nie istnieje	1.192093e-07	2.220446e-016

Wartości wyznaczonych liczb eta również okazały się identyczne z wynikiem funkcji nextfloat() dla każdego typu.

Liczba eta	Float16 (half)	Float32 (single)	Float64 (double)
Wyznaczony	6.0e-8	1.0e-45	5.0e-324
Julia- nextfloat(0.0)	6.0e-8	1.0e-45	5.0e-324

Liczba macheps (machine epsilon) jest najmniejszą liczbą dodatnią, która dodana do jedynki nie zostanie przez nią pochłonięta, natomiast ϵ (precyzja arytmetyki) stanowi górną granicą błędu zaokrągleń. Im wyższa liczba macheps, tym niższa precyzja obliczeń, czyli wyższe ϵ. Liczbe te są więc do siebie odwrotnie proporcjonalne.

Liczba eta to najmniejsza dodatnia liczba, jaką można zapisać w danej arytmetyce zmiennopozycyjnej. Liczba MIN_{SUB} to najmniejsza liczba zapisana w postaci subnormalnej. Dla standardu IEEE-754 obie te wartości są sobie równe.

MIN_{NOR} to minimum znormalizowane, minimalna wartość ze znormalizowaną mantysą. Funkcja floatmin() zwraca właśnie te wartości dla obu typów.

Float32	1.1754944e-38
Float64	2.2250738585072014e-308

Największe możliwe do zapisania wartości w poszczególnych arytmetykach zmiennopozycyjnych:

MAX	Float16 (half)	Float32 (single)	Float64 (double)
Wyznaczona	6.55e4	3.4011222e38	1.7967942882948848e308
Julia- floatmax()	6.55e4	3.4011222e38	1.7967942882948848e308
C- float.h	nie istnieje	3.402823466385e38	1.7976931348623157e308

Zadanie 2

Okazuję się, że teza Kahana $macheps = 3(\frac{4}{3} - 1) - 1$ nie jest prawdziwa dla wszystkich typów zmiennoprzecinkowych w języku Julia, natomiast wartość bezwzględna z funkcji Kahana jest już równa epsilonowi maszynowemu, co oznacza $macheps = \left|3(\frac{4}{3} - 1) - 1\right|$.

	Float16 (half)	Float32 (single)	Float64 (double)
$3(\frac{4}{3} - 1) - 1$	-0.000977	1.1920929e-7	-2.220446049250313e-16
$\left\|3(\frac{4}{3} - 1) - 1\right\|$	0.000977	1.1920929e-7	2.220446049250313e-16
Julia- eps()	0.000977	1.1920929e-7	2.220446049250313e-16

Zadanie 3

W arytmetyce double w standarcie IEEE 754 liczby zmiennopozycyjne są równomiernie rozmieszczone w przedziałach [0.5, 1.0], [1.0, 2.0] i [2.0, 4.0]. Dla każdego z nich, krok (δ) wynosi odpowiednio 2^{-53}, 2^{-52} i 2^{-51}. Poniżej przedstawiono wartości graniczne, oraz wartości o jeden krok mniejsze lub większe, zapisane w bitach z podziałem na cechę (wykładnik) oraz mantysę. Bit znaku celowo został pominięty, ponieważ w każdym z analizowanych przypadków jego wartość wynosi 0. Można zauważyć, że wzrost wartości liczby o jeden krok odpowiada zwiększeniu mantysy o 1. W momencie kiedy mantysa przepełni się, cecha zwiększa się o jeden, a mantysa zeruje się. Oznacza to koniec przedziału i rozpoczęcie nowego, w którym krok, czyli jeden bit mantysy będzie reprezentował dwa razy większą wartość niż w

przedziale poprzednim. Można z tego wysnuć wniosek, że każdy następny przedział jest dwukrotnie większy od poprzedniego.

	Reprezentacja bitowa	
Wartość	Cecha	Mantysa
0.5	01111111110	0000000000000000000000000000000000000000000000000000
0.5000000000000001	01111111110	0000000000000000000000000000000000000000000000000001
0.9999999999999999	01111111110	1111111111111111111111111111111111111111111111111111
1.0	01111111111	0000000000000000000000000000000000000000000000000000
1.0000000000000002	01111111111	0000000000000000000000000000000000000000000000000001
1.9999999999999998	01111111111	1111111111111111111111111111111111111111111111111111
2.0	10000000000	0000000000000000000000000000000000000000000000000000
2.0000000000000004	10000000000	0000000000000000000000000000000000000000000000000001
3.9999999999999996	10000000000	1111111111111111111111111111111111111111111111111111
4.0	10000000000	0000000000000000000000000000000000000000000000000000

Zadanie 4

Po 257736490 iteracjach udało się znaleźć najmniejszy x dla którego $x \times \frac{1}{x} \neq 1$. Dla $x = 1.000000057228997$ $x \times \frac{1}{x} = 0.9999999999999999$.

Zadanie 5

Iloczyn skalarny dwóch wektorów
$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$ i
$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$ wynosi
$-1.00657107000000 \cdot 10^{-11}$. Poniższe tabele przedstawiają sumy oraz błędy bezwzględne obliczone w dwóch precyzjach: Float32 i Float64 dla czterech algorytmów sumowania:

A. w przód,
B. w tył,
C. od największego do najmniejszego,
D. od najmniejszego do największego.

Float32		
Algorytm	Suma	Błąd
“w przód”	-0.3472038161853561	0.3472038161752904
“w tył”	-0.4543457	0.4543457
“od największego do najmniejszego”	-0.3472038161853561	0.3472038161752904
“od najmniejszego do największego”	-0.39291382	0.39291382

Float64		
Algorytm	Suma	Błąd
“w przód”	1.0251881368296672e-10	1.1258452438296672e-10
“w tył”	-1.5643308870494366e-10	1.4636737800494365e-10
“od największego do najmniejszego”	1.4636737800494365e-10	1.4636737800494365e-10
“od najmniejszego do największego”	1.4636737800494365e-10	1.4636737800494365e-10

Można zauważyć, że zarówno precyzja jak i algorytm mają duży wpływ na wyniki. Co oczywiste, najmniejszy błąd został uzyskany dla arytmetyki Float64.

Zadanie 6

Funkcje $f(x) = \sqrt{x^2+1} - 1$ oraz $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ są sobie równe, jednak dla niewielkich x dają różne wyniki. Tabela przedstawia wyniki funkcji dla wybranych $x = 8^{-n}$.

n	$f(x) = \sqrt{x^2+1} - 1$	$g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$
1	0.0077822185373186414	0.0077822185373187065
2	0.00012206286282867573	0.00012206286282875901
3	1.9073468138230965e-6	1.907346813826566e-6
5	4.656612873077393e-10	4.6566128719931904e-10
8	1.7763568394002505e-15	1.7763568394002489e-15
9	0.0	2.7755575615628914e-17
20	0.0	3.76158192263132e-37
50	0.0	2.4545467326488633e-91
100	0.0	1.204959932551442e-181
150	0.0	5.915260930833874e-272
178	0.0	1.6e-322
179	0.0	0.0
200	0.0	0.0

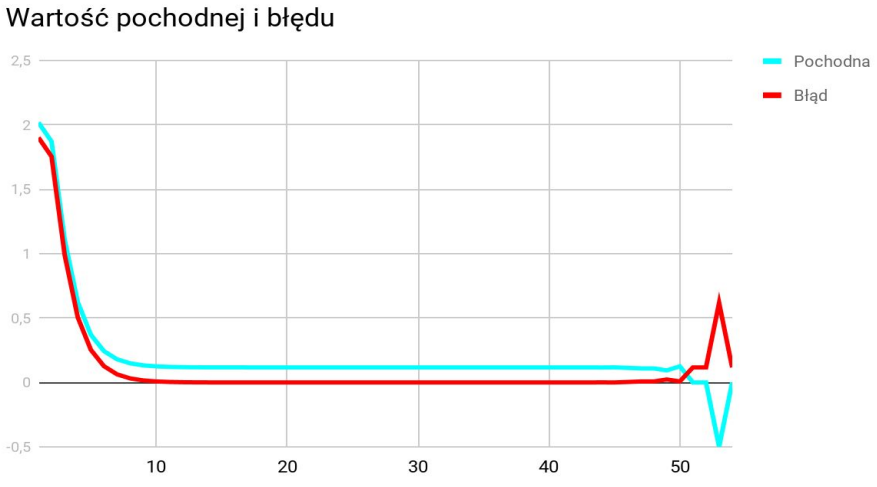
Można łatwo zauważyć, że $f(x)$ dużo szybciej zaczęła zwracać 0.0, mimo tego, że do $n = 8$ obie funkcje zwracały niemal identyczne wyniki. Odpowiedzialny za ten stan rzeczy jest problem pochłaniania. x^2 szybko staje się liczbą bardzo bliską 0.0. Przez to, dla $n \geq 9$ jest pochłaniane przez jedynkę pod pierwiastkiem, co z kolei sprawia, że wynikiem pierwiastka jest 1.0, a całej funkcji 0.0.

W przypadku $g(x)$ funkcja dużo dłużej zwraca poprawne wyniki. Kiedy x^2 zbliża się do zera, jest pochłaniane przez 1.0 pod pierwiastkiem, jednak do wyniku dodawane jest 1.0, a następnie x^2 jest dzielone przez 2. Tu jednak pochłanianie nie występuje i funkcja nie jest zerowana tak szybko.

Zadanie 7

Przybliżona wartość pochodnej w punkcie x wynosi $f'(x_0) \approx \tilde{f}'(x_0) = \frac{f(x_0+h)-f(x_0)}{h}$, gdzie $f'(x_0) = sinx + cos3x$, $x_0 = 1$ oraz $h = 2^{-n}$ dla $n = 0,\ 1,\ 2,\ ...,\ 54$. Błąd wynosi $\left| f'(x_0) - \tilde{f}'(x_0) \right|$. Dokładną wartość pochodnej można wyliczyć za pomocą wzoru $f'(x_0) = cosx - 3sin3x = 0.11694228168853815$. Poniższa tabela przedstawia wartości pochodnej i błędu dla n równego wielokrotności 6.

n	Wartość pochodnej	Wartość błędu
0	2.0179892252685967	1.9010469435800585
6	0.18009756330732785	0.0631552816187897
12	0.11792723373901026	0.0009849520504721099
18	0.11695767106721178	1.5389378673624776e-5
24	0.11694252118468285	2.394961446938737e-7
30	0.11694216728210449	1.1440643366000813e-7
36	0.116943359375	1.0776864618478044e-6
42	0.11669921875	0.0002430629385381522
48	0.09375	0.023192281688538152
54	0.0	0.11694228168853815



Można zauważyć, że wartość błędu spada wraz ze wzrostem n, jednak do pewnego momentu. Później wyniki załamują się. Najprawdopodobniej ma to związek z tym, że dla większych n, $h \approx 0.0$ zostaje pochłonięte przez $x_0 = 1.0$ w $x_0 + h$. Z tego powodu wynikiem licznika w $\tilde{f}'(x_0)$ i całej funkcji jest 0.0.