Obliczenia naukowe, sprawozdanie z listy 1

Data Paweł 250105

October 22, 2020

1 Zadanie

1.1 epsilon maszynowy

Liczba macheps *eps* (epsilon maszynowy) to najmniejsza liczba taka, że $eps > 0$, $fl(1 + eps) > 0$ i $fl(1 + eps) = 1 + eps$.

Liczbę *eps* wyznaczyłem iteracyjnie od 1, dzieliłem ją przez 2, jeśli nadal spełniała powyższe warunki.

Moje wyniki:

	Float16	Float32	Float64
Mój wynik	0.000977	1.1920929e-7	2.220446049250313e-16
Julia	0.000977	1.1920929e-7	2.220446049250313e-16
C float.h	-	1.1920929e-07	2.220446049250313e-16

Więc moje wyniki są dokładne, równe tym z języka Julia oraz C (w języku C nie ma zdefiniowanego *eps* 16 bitowego).

Liczba macheps jest dokładnie dwa razy większa od precyzji arytmetyki. Skoro precyzja arytmetyki to wartość, o którą maksymalnie tracimy dokładność danej liczby w czasie jej zapisu, to jest ona dwa razy mniejsza, niż najmiejsza wartość, która ma znaczenie przy dodawaniu/odejmowaniu.

1.2 Liczba eta

Liczba eta to najmniejsza liczba taka, że $eta > 0$. Liczę *eta* iteracyjnie od 1, dzielliłem ją przez 2, jeśli nadal spełniała powyższy warunek.

	Float16	Float32	Float64
Mój wynik	6.0e-8	1.0e-45	5.0e-324
Julia	6.0e-8	1.0e-45	5.0e-324

Liczba eta jest równa najmniejszej liczbie nieznormalizowanej (najmniejszej wartosci, jaką jesteśmy w stanie zapisać w danej arytmetyce).

1.3 Minimalna wartość

funkcja floatmin() zwraca najmniejszą znormalizowaną wartość w danej arytmetyce. W tabeli są wartości floatmin() dla typów float z języka Julia

Typ	floatmin
Float16	6.104e-5
Float32	1.1754944e-38
Float64	2.2250738585072014e-308

1.4 Maksymalna wartość

Wyznaczam największą wartość iteracyjnie zaczynając od liczby mniejszej o zera, która ma największą mantysę. Mnożę razy dwa dopóki nie spełnia warunku isinf()
Moje wyniki:

	Float16	Float32	Float64
Mój wynik	6.55e4	3.4028235e38	1.7976931348623157e308
Julia	6.55e4	3.4028235e38	1.7976931348623157e308

Wyniki są takie same.

2 Zadanie

Sprawdziłem tezę, czy da się obliczyć liczbę macheps formułą:

$$3(4/3 - 1) - 1$$

Wyniki:

	Float16	Float32	Float64
Mój wynik	0.000977	1.1920929e-7	2.220446049250313e-16
Julia	0.000977	1.1920929e-7	2.220446049250313e-16

Formuła poprawnie liczy epsilon maszynowy. Jedynie trzeba dodać wartość bezwzględną.

3 Zadanie

Zadanie polega na sprawdzeniu, ile jest liczb zmiennopozycyjnych w pewnym przedziale.
Zacznę od [1, 2]. Wyświetlę liczbę 1, kilka liczb większych od niej, korzystając z funkcji nextfloat(), liczbę 2 oraz mniejszą od niej przy pomocy prevfloat()
Oto wynik programu:

- One : 00111111111100000000000000000000000000000000000000000000000000000
- next : 00111111111100000000000000000000000000000000000000000000000000001
- next : 00111111111100000000000000000000000000000000000000000000000000010
- next : 00111111111100000000000000000000000000000000000000000000000000011
- Two : 01000000000000000000000000000000000000000000000000000000000000000
- prev : 00111111111111111111111111111111111111111111111111111111111111111

Można zauważyć, że funkcje nextfloat() i prevfloat() zmieniają mantysę liczby o 1. Więc dla tej samej cechy (czyli dla liczb z przedziału $[2^x, 2^{x+1}]$ dla pewnego x) istnieje tyle samo liczb (2^{52}).
Zatem przedziały $[\frac{1}{2}, 1]$, $[1, 2]$, $[2, 4]$ są równoliczne. Krok wynosi odpowiedno 2^{-53}, 2^{-52}, 2^{-51}.

4 Zadanie

Mam znaleźć najmniejszą liczbę Float64 $1 < x < 2$ taką, że $x * \frac{1}{x} \neq 1$. Sprawdzając w pętli ten warunek, zaczynając od jedynki, dostaję liczbę

1.000000057228997

Według standardowej matematyki obe strony równania są równe, w trakcie obliczeń maszynowych liczby są zaokrąglane, przez co dostajemy niedokładne wyniki.

5 Zadanie

Zadanie polega na policzeniu iloczynu skalarnego dwoch wektorów na 4 różne sposoby.

1. suma "w przód", czyli dodajemy wartości w kolejności, w jakiej występują w tablicy
2. suma "w tył", czyli dodajemy w odwrotnej kolejności, niż w punkcie 2
3. suma od największej do najmniejszej wartości (suma dodatnich od największej, suma ujemnych od najniższej, suma obu podsum)
4. odwrotnie niż w punkcie 3

W tabeli znajdują się wyniki

sposób	Float32	Float64
1	-0.2499443	1.0251881368296672e-10
2	-0.2043457	-1.5643308870494366e-10
3	-0.25	0.0
4	-0.25	0.0

Porównując z prawdziwym wynikiem $-1.00657107000000 * 10^{-11}$ najbliżej jest sposób pierwszy dla Float64, czyli suma wartości od początku tablicy. Float32 jest gorszy w każdym sposobie przez krótszą mantysę.

To zadanie pokazuje, że w arytmetyce IEEE-754 dodawanie nie jest przemienne.

6 Zadanie

Mam policzyć wartości dla podanych funkcji:

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

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

Dla $x = 8^{-1}$, 8^{-2}, 8^{-3}, ...

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

Jak widać w tabeli, wyniki są różne, mimo tego, że te funkcje są matematycznie takie same. Im wartość x jest mniejsza, bym są większe różnice między wynikami. Funkcja f dla wartości 8^{-9} wynosi 0, co może być spowodowane odejmowaniem podobnych wartości.

Zatem funkcja g zwraca dokładniejsze wartości.

7 Zadanie

Istnieje wzór na przyblizenie pochodnej funkcji

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

Sprawdzę różnicę pochodnej funkcji

$$f(x) = sinx + cos3x$$

oraz jej przybliżenia za pomocą powyższego wzoru dla
$h = 2^{-n}$., n = 0, 1, 2, ..., 54.

h	$\lvert\widetilde{f}'(x) - f'(x)\rvert$
2^{-0}	1.9010469435800585
2^{-1}	1.753499116243109
2^{-2}	0.9908448135457593
2^{-3}	0.5062989976090435
2^{-4}	0.253457784514981
2^{-5}	0.1265007927090087
2^{-6}	0.0631552816187897
2^{-7}	0.03154911368255764
2^{-8}	0.015766832591977753
2^{-9}	0.007881411252170345
2^{-10}	0.0039401951225235265
2^{-11}	0.001969968780300313
2^{-12}	0.0009849520504721099
2^{-13}	0.0004924679222275685
2^{-14}	0.0002462319323930373
2^{-15}	0.00012311545724141837
2^{-16}	6.155759983439424e-5
2^{-17}	3.077877117529937e-5
2^{-18}	1.5389378673624776e-5
2^{-19}	7.694675146829866e-6
2^{-20}	3.8473233834324105e-6
2^{-21}	1.9235601902423127e-6
2^{-22}	9.612711400208696e-7
2^{-23}	4.807086915192826e-7
2^{-24}	2.394961446938737e-7
2^{-25}	1.1656156484463054e-7
2^{-26}	5.6956920069239914e-8
2^{-27}	3.460517827846843e-8
2^{-28}	4.802855890773117e-9
2^{-29}	5.480178888461751e-8
2^{-30}	1.1440643366000813e-7
2^{-31}	1.1440643366000813e-7
2^{-32}	3.5282501276157063e-7
2^{-33}	8.296621709646956e-7
2^{-34}	8.296621709646956e-7
2^{-35}	2.7370108037771956e-6

2^{-36}	1.0776864618478044e-6
2^{-37}	1.4181102600652196e-5
2^{-38}	1.0776864618478044e-6
2^{-39}	5.9957469788152196e-5
2^{-40}	0.0001209926260381522
2^{-41}	1.0776864618478044e-6
2^{-42}	0.0002430629385381522
2^{-43}	0.0007313441885381522
2^{-44}	0.0002452183114618478
2^{-45}	0.003661031688538152
2^{-46}	0.007567281688538152
2^{-47}	0.007567281688538152
2^{-48}	0.023192281688538152
2^{-49}	0.008057718311461848
2^{-50}	0.11694228168853815
2^{-51}	0.11694228168853815
2^{-52}	0.6169422816885382
2^{-53}	0.11694228168853815
2^{-54}	0.11694228168853815

Jak widać w tabeli, od $n = 0$ do $n = 28$ wartość przybliżona jest coraz dokladniejsza, lecz dla większych n różnica robi się większa.

Powodem jest fakt, że dla małego h zachodzi $x + h = x$.