Obliczenia Naukowe
lista 1

Mateusz Tofil

24 października 2021

1 Zadanie 1

1.1 Opis zadania

W tym zadaniu należało wyznaczyć *macheps* czyli najmniejszą liczbę macheps, która spełnia następująca nierówność $fl(1.0+macheps) > 1.0$. Następnie trzeba było wyznaczyć liczę *eta*, czyli najmniejszą liczbę większą od 0 tj. $eta > 0.0$. Kolejnym zadaniem, było wyznaczenie największej liczby *max*. Każdą z tych liczb (czyli. *macheps, eta, max*) należało wyznaczyć itercyjnie dla wszystkich typów zmiennopozycyjnych.

1.2 Metoda rozwiązania

W pliku `zad1.jl` oraz w `zad1floath.c` znajdują się programy, z których otrzymałem wyniki przeprowadzonych badań. W każdym z podproblemie zasada działania była bardzo podobna i polegała na dzieleniu lub mnożeniu liczby początkowej, aż do momentu kiedy nie zostało spełnione zdanie logiczne.

Badałem liczbę *macheps* w pętli, aż do momentu kiedy nie zaszedł warunek $1 + current/2 \neq 1$. W momencie spełnienia warunku, pętla kończyła swoją pracę i zwraca aktualna wartość dla której warunek zachodzi, czyli *macheps*.

Analogiczny algorytmy wykorzystałem do wyznaczenia liczby *eta*. W każdej iteracji dzieliłem liczbę do momentu kiedy nie zaszedł warunek $current/2 \neq 0$.

Licząc liczbę maksymalną zatrzymujemy się po osiągnięciu nieskończoności, która w rzeczywistości została przekroczona. Po wyjściu z pętli należy dodawać do liczby połowy przerwy między nieskończonością. Operacje powtarzać, aż do momentu gdy $\frac{x}{2^k} \geq 1$ dla pewnego k.

1.3 Otrzymane wyniki

W tabelach o numerach 1, 2, 3 zaprezenotwałem zestawienia wyników, które otrzymałem z przeprowadzonych przeze mnie badań. Porównuje je z budowanymi funkcjami w języku Julia, takimi jak np. *eps*() czy $nextfloat()$) Otrzymane wyniki są zgodnę z wbudowanymi funkcjami, co jednoznacznie stwierdza, że napisane przeze mnie funkcje są poprawnie napisane i zwracają poprawne wyniki.

typ	moja funkcja	funkcja *eps*()	float.h
Float16	0.000977	0.000977	b.d.
Float32	1.1920929e-7	1.1920929e-7	1.1920928955e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.2204460493e-16

Tablica 1: Wartości epsilona maszynowego dla typów zmiennopozycyjnych

typ	moja funkcja	funkcja $nextfloat(0.0)$
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Tablica 2: Wartości *eta* dla typów zmiennopozycyjnych

typ	moja funkcja	funkcja $nextfloat(0.0)$
Float16	6.55e4	6.55e4
Float32	3.4028235e38	3.4028235e38
Float64	1.7976931348623157e308	1.7976931348623157e308

Tablica 3: Wartości max dla typów zmiennopozycyjnych

1.3.1 Macheps a precyzja arytmetyki

Z wykładu wiemy, że precyzja arytmetyki to $\frac{1}{2}\beta^{1-t}$. Dla typu pojedycznej prezycji przeznaczone jest 24-bity, natomiast dla podwójnej już 53-bity. Podstawiając, dane to wzoru wyżej, otrzymujemy wyniki, z których widzimy, że prezyja arytmetyki jest dwa razy większa od macheps.

typ	precyzja arytmetyki	macheps
Float32	5.960464477539063e-8	1.1920929e-7
Float64	1.1102230246251565e-16	2.220446049250313e-16

Tablica 4: Porównanie prezyzji arytmetyki do macheps

1.3.2 *eta* a liczba MIN_{sub}

Liczba *eta* i liczba MIN_{sub} leżą w tym samym rzędzie, a różnica między nimi jest bardzo mała.

1.3.3 Związek między `floatmin()`, a liczbą MIN_{nor}

Podobnie jak liczby *eta* i MIN_{sub}, wartości zwracane przez funkcje `floatmin()` leżą w tym samym rzędzie co liczba MIN_{nor}

typ	funkcja $floatmin()$	MIN_{nor}
Float32	1.1754944e-38	1.2e-38
Float64	2.2250738585072014e-308	2.2e-308

Tablica 5: Porównanie wartości floatmin() i MIN_{nor}

1.4 Wnioski

Liczby w komputerze zgodne ze standardzem IEEE 754 mają skończoną precyję, co sprawia że ma skończone zakresy do reprezentacji liczb.

2 Zadanie 2

2.1 Opis zadania

Zadanie to polegało na sprawdzeniu, czy jesteśmy w stanie obliczając wartość wyrażenia

$$3 * (\frac{4}{3} - 1) - 1$$

otrzymać wartość epsilona maszynowego.

2.2 Metoda rozwiązania

Napisałem 3 funkcję, dla każdego typu `Float16, Float32, Float64`, która obliczała wyżej wymienione wyrażenie.

2.3 Otrzymane wyniki

Otrzymane wyniki porównałem z wcześniejszymi wynikami z poprzednich zadań i zestawiłem w tabeli poniżej.

typ	wynik wyrażenia	*eps*
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Tablica 6: Porównanie wartości z wyrażenia a) z *eps*

Jak widać część naszych wyników pokrywają się. W miejsach gdzie wyniki nie zgadzają się, można zauważyć, są to liczby przeciwne. Najprawdopodobniej spowodowane jest to tym że liczba $\frac{4}{3}$ w rozwinięciu binarnym ma nieskończone rozwinięcie. Rozwinięcie to prezentuje się następująco 1.(10). Liczba bitów znaczących dla typów danych wynosi:

- `Float16` - 10 bitów
- `Float32` - 23 bity
- `Float64` - 52 bity

Więc dla typu Float16 i Float64 ostatnią cyfrą mantysy jest 0, w przeciwieństwie do typu Float32, gdzie ostatnia cyfra mantysy to 1. To właśnie decyduje o znaku odejmowania.

2.4 Wnioski

Żyjąc w świecie gdzie istnieje tylko skończona dokłądność reprezentacji, niektóre równania dające w matematyce tożsamości, w arytmetyce zmiennoprzecinkowej mogą dawać zupełnie różne wyniki.

3 Zadanie 3

3.1 Opis problemu

W tym zadaniu, problem z jakim musiałem się zmierzyć to było zbadanie rozmieszczenia liczb zmiennoprzecinkowych w arytmetyce IEEE 754 o podwójnej prezycji. Rozmieszczenie liczb należało przebadać na różnych przedziałach liczbowych.

3.2 Metoda rozwiązania

Program do tego zadania znajduję się w pliku `zad3.jl`. Do problemu można było podejść w sposób iteracyjny sprawdzjąc czy odstęp między liczbami w przedziale [1,2] wyniosi 2^{-52}, natomiast to rozwiązanie jest bardzo długie. Można szybciej, korzystając z wbudowanej funckji `bitstring` musimy porównań mantysy dwóch liczb podanych na wejściu. Jeżeli mantysy, są równe sobie to na tym przedziale, rozmieszczenie pomiędzy nimi jest równomierne w każdym miejscu. W przeciwnym razie rozmieszczenie jest nierównomierne. Dla arytmetyki `Float64`, bias to 1023, a mantysa to 52 bity. Zatem aby sprawdzić jak bardzo zmienia się liczba, należy zwrócić uwagę co dzieje się z mantysą, po dodaniu jedynki.

$$2^{cecha-bias} * 2^{-mantysa}$$

3.3 Otrzymane wyniki

Liczby w przedziale $[1, 2]$ są rozmieszone równomiernie co 2^{-52}. Dodatkowo wyznaczyłem odlgełości między liczbami dla innych przedziałów, wyniki zapisałem w tableki poniżej.

przedział	odległośći między liczbami
$[0.5, 1]$	1.1102230246251565e-16
$[1, 2]$	2.220446049250313e-16
$[2, 4]$	4.440892098500626e-16

Tablica 7: Przedziały i odległości między liczbami w danym przedziale

3.4 Wnisoki

Liczby w IEEE 754 są reprezntoane z określoną dokładnością różniącą się zależnie od przedziału, w którym się znajdują, przedziały bliżej zera są bardziej gęstrze niż przedziały oddalone dalej.

4 Zadanie 4

4.1 Opis problemu

Problem, jaki znajdował się w tym zadaniu to było znalezienie dwóch liczb zmiennoprzecinkowych. Pierwsza liczba to była liczba x leżąca w przedziale $1 < x < 2$, taka, że spełnia nierówność:

$$x * \frac{1}{x} \neq 1$$

Druga część opierała się na podobnnym zadaniu, tylko tym razem należało znaleźć najmniejszą taką liczbę, która spełnia powyższą nierówność.

4.2 Metoda rozwiązania

Rozwiązania znajdują sie w pliku `zad4.jl`. Metodologia polega na zaczęciu od liczby, która jest dolnym ograniczeniem powyższych nierówności i powiększaniu jej do momentu gdy nierówność nie będzie prawdziwa. Podpunkt ten rozbiłem na dwa mnijesze jeszcze podpunkty, ze względu na najmniejszą liczbę dodatnią (b1), czy najmniejszą liczbę w ogólności. (b2)

4.3 Otrzymane wyniki

Napisane przez mnie algorytmy, zwróciły następujące wyniki:

podpunktu	szukany x	wartość wyrażenia x * (1/x)
a	1.000000057228997	0.9999999999999999
$b1$	1.0e-323	inf
$b2$	-1.0e-323	inf

Tablica 8: Szukane wartości x i wartości wyrażenia

4.4 Wnioski

Liczby reprezentowane w komputerze mają skończoną prezycję i chcąc policzyć nawet bardzo łatwe (nie)równości z matematyki, komputer ma problemy z poprawnym obliczaniem. Podpunkt b dał zaskakujące wyniki, mianowicie najmniejsza liczba dodatnia i najmnijesza liczba w ogólności są równe co do wartości bezwzględnej.

5 Zadanie 5

5.1 Opis problemu

Zadanie 5 polega na eksperymentalnym obliczeniu iloczynów skalarnych dwóch wektorów, które na wejściu są już podane. Eksperymenty należało przeprowadzić na 4 różnych algorytmach liczenia iloczy skalarnego:

(a) liczenie w przód, zaczynając od początku tablicy

(b) liczenie w tył, zaczynając od końca tablicy

(c) liczenie od największego iloczynu do najmniejszego

(d) liczenie od najmniejszego iloczyny do największego

5.2 Metoda rozwiązania

Zaimplementowane algorytmy znajdują się w pliku `zad5.jl`. Reprezentują wcześniej metody rozwiązane w języku Julia.

5.3 Otrzymane wyniki

podpunkt	suma
a	1.0251881368296672e-10
b	-1.5643308870494366e-10
c	0.0
d	0.0

Tablica 9: Wyniki dla `Float64`

podpunkt	suma
a	-0.3472038161853561
b	-0.3472038162872195
c	-0.5
d	-0.5

Tablica 10: Wyniki dla `Float32`

5.4 Wnioski

Z punkty widzenia matematyki, wykonywanie tych algorytmów powinno dać równe wyniki, bez względu na kolejność wykonywania działać. (Zachowując reguły kolejnośći wykonywania działań). Natomisat w komputerze, kolejność wykonywanych działań może zmienić się w zależności od wykonywanych działań.

6 Zadanie 6

6.1 Opis problemu

W tym zadaniu należało obliczyć wartości dwóch równoważnych funkcji (pod względem matematycznym) dla argumentów 8^{-1}, 8^{-2}, 8^{-3}... 8^{-k} $k \in \mathbb{N}$. Równoważne funckje to:

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

6.2 Metoda rozwiązania

Zaimplementowane algorytmy znajdują się w pliku `zad6.jl`. Reprezentują wcześniej metody rozwiązane w języku Julia.

6.3 Otrzymane wyniki

Jak łatwo zauważyć, funckja $f(x)$ szybko osiąga wartości równe 0.0, gdy funckja $g(x)$ wciąż liczy dla mniejszych argumentów wartości funkcji. Funkcja $g(x)$ jest 10-razy efektywniejsza niż funkcja $f(x)$.

6.4 Wnioski

Podobnie jak w zadaniu poprzednim, mimo że dwie funkcje, $f(x)$ i $g(x)$ matematycznie są równoważne, komputer zwraca prawidłowe wyniki tylko do pewnego momentu, a w ostateczności zwraca niepoprawne.

7 Zadanie 7

7.1 Opis problemu

W tym zadaniu należało wyliczyć wartość pochodnej $f(x)$ za pomocą wzoru

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h}, h \to 0$$

i porównać ją z wartością matematycznej pochodnej funckji w punkcie x_0

$$f(x) = sin(x) + cos(3x)$$
$$f'(x) = cos(x) - 3cos(3x)$$

k	$f(8^{-k})$	$g(8^{-k})$
1	0.0077822185373186414	0.007782218537318706
2	0.00012206286282867573	0.00012206286282875901
3	1.9073468138230965e-6	1.907346813826566e-6
4	2.9802321943606103e-8	2.9802321943606116e-8
5	4.656612873077393e-10	4.6566128719931904e-10
6	7.275957614183426e-12	7.275957614156956e-12
7	1.1368683772161603e-13	1.1368683772160957e-13
8	1.7763568394002505e-15	1.7763568394002489e-15
9	0.0	2.7755575615628914e-17
10	0.0	4.336808689942018e-19
10	0.0	4.336808689942018e-19
20	0.0	3.76158192263132e-37
30	0.0	3.2626522339992623e-55
40	0.0	2.8298997121333476e-73
50	0.0	2.4545467326488633e-91
60	0.0	2.1289799200040754e-109
70	0.0	1.8465957235571472e-127
80	0.0	1.6016664761464807e-145
90	0.0	1.3892242184281734e-163
100	0.0	1.204959932551442e-181
110	0.0	1.0451361413042083e-199
120	0.0	9.065110999561118e-218
130	0.0	7.862730431637126e-236
140	0.0	6.819831532519088e-254
150	0.0	5.915260930833874e-272
160	0.0	5.1306710016229703e-290
170	0.0	4.450147717014403e-308
180	0.0	0.0
190	0.0	0.0
200	0.0	0.0

Tablica 11: Porównanie wartości funkcji f i g z zadania 6

7.2 Metoda rozwiązania

Zaimplementowane algorytmy znajdują się w pliku `zad7.jl`. Reprezentują wcześniejsze implementacje równań.

7.3 Otrzymane wyniki

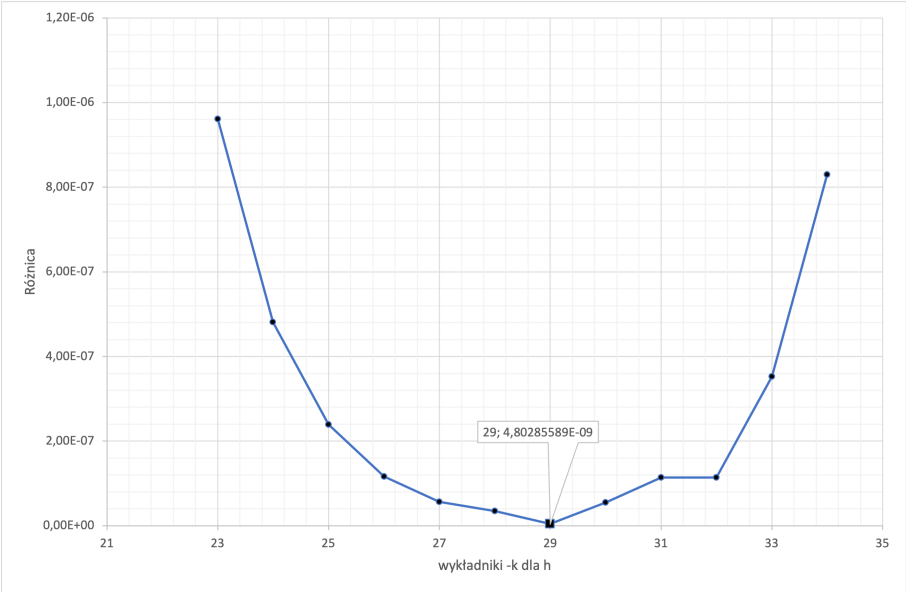
Wartość pochodnej w punkcie x + 0 = 1 to 0.11694228168853815. Z otrzymanych wyników możemy wywnioskować, że najlepsze przybliżenie wartośći pochodnej w tym punkcie jest dla $h = 2^{-28}$, gdzie błąd jest najmniejszy. Po tej wartości błąd rośnie, a dla $h = 2^{-54}$ wynosi już prawie 0.117.

h	h+1	$\widetilde{f}'(x_0)$	$\lvert f'(x_0) - \widetilde{f}'(x_0)\rvert$
2^{-0}	2.0	2.0179892252685967	1.9010469435800585
2^{-1}	1.5	1.8704413979316472	1.753499116243109
2^{-2}	1.25	1.1077870952342974	0.9908448135457593
2^{-3}	1.125	0.6232412792975817	0.5062989976090435
2^{-4}	1.0625	0.3704000662035192	0.253457784514981
2^{-5}	1.03125	0.24344307439754687	0.1265007927090087
⋮	⋮	⋮	⋮
2^{-22}	1.000000238418579	0.11694324295967817	9.612711400208696e-7
2^{-23}	1.0000001192092896	0.11694276239722967	4.807086915192826e-7
2^{-24}	1.0000000596046448	0.11694252118468285	2.394961446938737e-7
2^{-25}	1.0000000298023224	0.116942398250103	1.1656156484463054e-7
2^{-26}	1.0000000149011612	0.11694233864545822	5.6956920069239914e-8
2^{-27}	1.0000000074505806	0.11694231629371643	3.460517827846843e-8
2^{-28}	1.0000000037252903	0.11694228649139404	4.802855890773117e-9
2^{-29}	1.0000000018626451	0.11694222688674927	5.480178888461751e-8
2^{-30}	1.0000000009313226	0.11694216728210449	1.1440643366000813e-7
2^{-31}	1.0000000004656613	0.11694216728210449	1.1440643366000813e-7
2^{-32}	1.0000000002328306	0.11694192886352539	3.5282501276157063e-7
2^{-33}	1.0000000001164153	0.11694145202636719	8.296621709646956e-7
2^{-34}	1.0000000000582077	0.11694145202636719	8.296621709646956e-7
2^{-35}	1.0000000000291038	0.11693954467773438	2.7370108037771956e-6
⋮	⋮	⋮	⋮
2^{-49}	1.0000000000000018	0.125	0.008057718311461848
2^{-50}	1.0000000000000009	0.0	0.11694228168853815
2^{-51}	1.0000000000000004	0.0	0.11694228168853815
2^{-52}	1.0000000000000002	-0.5	0.6169422816885382
2^{-53}	1.0	0.0	0.11694228168853815
2^{-54}	1.0	0.0	0.11694228168853815

Tablica 12: Przybliżone wartości pochodnej

7.4 Wnioski

Liczby zmiennoprzecinkowe bliskie zeru posiadają niewielką liczbę cyfr znaczących w swoim zapisie. Z każdą iteracją, tracona jest dokładność obliczeń, aż do momentu w którym je odrzucamy. Zatem należy uważać, pracując na liczbach bliskim zeru.



Rysunek 1: Wykres