Obliczenia naukowe - lista pierwsza

Bartosz Rajczyk

15 października 2019

Spis treści

1 Zadanie 1.

1.1 Opis problemu

Problem polega na wyznaczeniu liczb:

1. *macheps*, czyli najmniejszej liczby spełniającej $fl(1.0 + macheps) > 1.0$
2. *eta*, czyli najmniejszej reprezentowalnej liczby
3. *max*, czyli największej reprezentowalnej liczby

dla typów zmiennoprzecinkowych o precyzji 16-, 32- oraz 64-bitowej.

1.2 Rozwiązanie

Zaprezentowane rozwiązania w pliku `1.jl` polegają na iteracyjnym dzieleniu lub mnożeniu liczby początkowej (uzyskiwanej najczęściej przez wywołanie `one(type)` zwracające jedynkę w podanym typie danych w Julii) aż do spełnienia określonego warunku, będącego w kolejności:

1. $1 + aktualna\_liczba/2 \neq 1$
2. $aktualna\_liczba/2 \neq 0$
3. $aktualna\_liczba * 2 \neq \infty$

Przy spełnieniu tego warunku pętla jest zakańczana i zwracana jest aktualna liczba.

1.3 Wyniki i interpretacja

typ danych	**moja funkcja**	**eps**	**float.h**
Float16	0.000977	0.000977	niedostępne
Float32	1.1920929e-7	1.1920929e-7	1.192093e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.220446e-16

Tabela 1: wartości epsilona maszynowego

typ danych	**moja funkcja**	**nextfloat(0)**
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Tabela 2: wartości eta

typ danych	moja funkcja	floatmax
Float16	3.277e4	6.55e4
Float32	1.7014118e38	3.4028235e38
Float64	8.98846567431158e307	1.7976931348623157e308

Tabela 3: wartości max

Z powyższych danych wynika, że zaimplementowane przeze mnie funkcje poprawnie zwracały wyniki poza przypadkiem wyliczania liczby maksymalnej, gdzie uzyskane rezultaty były dwukrotnie za małe. Wynika to najprawdopodobniej z tego, że mnożąc liczbę za każdym krokiem dwukrotnie wkraczamy w pewnym momencie na dokładną wartość nieskończoności, kiedy poprawnym rozwiązaniem jest jedna liczba przed nią, stąd ta rozbieżność.

1.4 Wnioski

Standardowa reprezentacja liczb zgodna z IEEE754 w komputerze ma skończoną dokładność i posiada skończone zakresy liczb, które potrafi reprezentować. Jednocześnie istnieją dwie osobne i rozróżnialne reprezentacje bardzo małych liczb - znormalizowane i zdenormalizowane. Wyznaczone przez nas liczby eta oraz liczby zwracane przez $nextfloat(0)$ dla różnych typów danych maja postać zdenormalizowaną (aczkolwiek nie pokrywają się idealnie z wartościami podawanymi przez źródła [1], najpewniej przez zaokrąglenia). Inne funkcje, jak chociażby `floatmin` zwracają wartości znormalizowane. Należy więc pamiętać o istnieniu obu reprezentacji i odpowiednim używaniu ich - także ze względu na używany procesor, bo ze źródeł wynika, że nie każde CPU obsługuje liczby zdenormalizowane całkowicie poprawnie.

2 Zadanie 2.

2.1 Opis problemu

Problem polega na odnalezieniu epsilonu maszynowego używając wyrażenia:

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

w arytmetyce zmiennoprzecinkowej.

2.2 Rozwiązanie

Zaprezentowane rozwiązanie w pliku `2.jl` polega na użyciu funkcji `one` do uzyskania wartości 1 w danym typie liczbowym, a nastepnie wykonanie podanego działania korzystając z liczb opartych na tej jedynce.

2.3 Wyniki i interpretacja

typ danych	**moja funkcja**	**eps**
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Tabela 4: obliczone wartości 3*(4/3 - 1) - 1

Widzimy, że wartość bezwzględna z uzyskanych przez nas wyników pokrywa się z epsilonem zwracanym przez funkcję. Zmiana znaku spowodowana jest najpewniej tym, że liczba bitów znaczących dla kolejnych danych wynosi:

- Float16 - 10
- Float32 - 23
- Float64 - 52

A rozwinięciem binarnym ułamka 4/3 jest 1.(10), więc dla typów Float16 i Float52 ostatnią cyfrą mantysy będzie 0, a dla typu Float32 - 1, co powinno decydować o znaku odejmowania.

2.4 Wnioski

Przez skończoną dokładność reprezentacji, niektóre równania dające w normalnej arytmetyce zero, w arytmetyce zmiennoprzecinkowej mogą dawać inne wyniki.

3 Zadanie 3.

3.1 Opis problemu

Problem polega na sprawdzeniu rozmieszczenia liczb zmiennoprzecinkowych w arytmetyce IEEE 754 podwójnej precyzji w danych przedziałach liczbowych.

3.2 Rozwiązanie

Zaprezentowane rozwiązania w pliku `3.jl` to dwa różne podejścia do tego problemu. Pierwsze polega na ręcznym przeiterowaniu przez wszystkie istniejące liczby podanego przedziału i sprawdzenie, czy odległości między kolejnymi rzeczywiście zawsze wynoszą sprawdzaną wartość. Drugie to bardziej analityczne rozwiązanie bazujące na naszej znajomości standardu IEEE754 i analizie rozmieszczenia bitów zwracanego przez funkcję `bitstring`. W tym podejściu porównujemy pierwszą i ostatnią liczbę z przedziału; jeżeli ich eksponenty są inne, wyklucza to równomierny rozkład pomiędzy nimi. Jeżeli są równe, możemy obliczyć jak bardzo zmienia się liczba przy powiększeniu o jeden mantysy używając wzoru:

$$2^{eksponenta-1023} * 2^{-52}$$

ponieważ biasem dla eksponenty Float64 jest 1023, a mantysa ma 52 bity znaczące.

3.3 Wyniki i interpretacja

Utworzona przez mnie funkcja potwierdziła, że w przedziale $[1, 2]$ liczby rozmieszczone są co 2^{-52}. Dodatkowo udało się wyznaczyć rozmieszczenie liczb dla innych przedziałów:

przedział	**odległości**
[0.5, 1]	1.1102230246251565e-16
[1, 2]	2.220446049250313e-16
[2, 4]	4.440892098500626e-16

Tabela 5: przedziały i odległości między liczbami

Te wyniki pokrywają się z moim rozumieniem standardu IEEE754 - ze względu na powiększenie się eksponenty o jeden, odległości pomiędzy kolejnymi liczbami rosną dwukrotnie (ponieważ eksponenta jest wykorzystywana jako wykładnik w $2^{eksponenta}$).

3.4 Wnioski

Liczby w IEEE754 są reprezentowane z określoną dokładnością różniącą się zależnie od przedziału, w którym się znajdują.

4 Zadanie 4.

4.1 Opis problemu

Problem polega na znalezieniu dwóch liczb:

1. najmniejszego $1 < x < 2$, takiego że $1 * (1/x) \neq 1$
2. najmniejszego $0 < x$, takiego że $1 * (1/x) \neq 1$

4.2 Rozwiązanie

Zaprezentowane rozwiązania w pliku `4.jl` polegają na rozpoczęciu od jednej liczby powyżej dolnego ograniczenia dla x i powiększaniu jej do następnej aż do uzyskania szukanej nierówności.

4.3 Wyniki i interpretacja

// TODO

4.4 Wnioski

// TODO

5 Zadanie 5.

5.1 Opis problemu

Problem polega na obliczaniu iloczynu skalarnego dwóch podanych wektorów na kilka róznych sposobów:

1. w przód, zaczynając dodawanie od pierwszych indeksów
2. w tył, zaczynając dodawanie od ostatnich indeksów
3. od największego iloczynu
4. od najmniejszego iloczynu

5.2 Rozwiązanie

Zaprezentowane rozwiązania w pliku `5.jl` polegają na dosłownej implementacji podanych algorytmów.

5.3 Wyniki i interpretacja

algorytm	**wynik**
1	1.0251881368296672e-10
2	-1.5643308870494366e-10
3	0.0
4	0.0

Tabela 6: Wyniki dla Float64

algorytm	**wynik**
1	-0.3472038161853561
2	-0.3472038162872195
3	-0.5
4	-0.5

Tabela 7: Wyniki dla Float32

5.4 Wnioski

Kolejność wykonywanych obliczeń może drastycznie wpłynąć na otrzymywany wynik.

6 Zadanie 6.

6.1 Opis problemu

Problem polega na obliczeniu wartości dwóch równoważnych matematycznie funkcji w dla kolejnych wartości $8^{-1}, 8^{-2}, 8^{-3}, ...$. Funkcje to:

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

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

6.2 Rozwiązanie

Zaprezentowane rozwiązania w pliku `6.jl` polegają na dosłownej implementacji podanych równań.

6.3 Wyniki i interpretacja

wartość x	**f**(8^{-x})	**g**(8^{-x})
1	0.0077822185373186414	0.0077822185373187065
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
20	0.0	3.76158192263132e-37
40	0.0	2.8298997121333476e-73
60	0.0	2.1289799200040754e-109
80	0.0	1.6016664761464807e-145
100	0.0	1.204959932551442e-181
120	0.0	9.065110999561118e-218
140	0.0	6.819831532519088e-254
160	0.0	5.1306710016229703e-290
180	0.0	0.0

Tabela 8: porównanie wyników f i g

Widzimy, że fukcja f bardzo szybko daje wyniki równe zeru, natomiast funkcja g pozwala obliczyć swoją wartość bardzo blisko minimalnych liczb w zakresie typu danych Float64. Widzimy dodatkowo, że dla wartości x, dla których obie funkcje nadal dają rezultaty, są one dosyć podobne, chociaż nie identyczne. Funkcja f radzi sobie gorzej ze względu na to, że operuje ona na wartościach bardzo bliskich zeru ze względu na odejmowanie jedynki od pierwiastka - w ten sposób traci dużo cyfr znaczących z wyniku pierwiastka. Funkcja g obchodzi ten problem i dlatego pozwala na otrzymanie wyników przy znacznie mniejszym wejściu.

6.4 Wnioski

Należy wykonywać obliczenia w ten sposób, aby liczba cyfr znaczących przy kolejnych działaniach zbytnio się nie różniła, ponieważ pozwala to na znaczne poprawienie dokładności obliczeń.

7 Zadanie 7.

7.1 Opis problemu

7.2 Rozwiązanie

7.3 Wyniki i interpretacja

7.4 Wnioski

8 Bibliografia

Literatura

[1] W. Kahan *Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic*. 1997