

Wprowadzenie

Poniższy dokument jest sprawozdaniem z listy 1 z przedmiotu Obliczenia Naukowe

1 Zadanie 1

Zadanie 1 ma na celu rozpoznanie arytmetyki.

1.1 Epsilon maszynowy

Pierwszym podpunktem zadania było napisać program w języku Julia, który wyznacza iteracyjnie epsilony maszynowe dla typów zmiennoprzecinkowych: Float16, Float32, Float64. Epsilon to najmniejsza liczba > 0.0 , która maszynowo następuje po 1.0. Zadanie zostało wykonane poprzez dzielenie przez 2 w pętli potencjalnego epsilon i sprawdzanie czy dodanie go do jedynki będzie skutkowało zwiększeniem wyniku. Oto output programu, porównujący wyznaczone doświadczalnie epsilony do wartości zwracanych przez funkcje `eps(Float16)`, `eps(Float32)`, `eps(Float64)`.

```
Float16
iteracyjnie: 0.000977
funkcja eps: 0.000977
```

```
Float32
iteracyjnie: 1.1920929e-7
funkcja eps: 1.1920929e-7
```

```
Float64
iteracyjnie: 2.220446049250313e-16
funkcja eps: 2.220446049250313e-16
```

Wyniki doświadczenia są za każdym razem precyzyjne i zgodne z wyjściem funkcji `eps()`. Porównując to z danymi zawartymi w pliku `float.h`:

```
FLT_EPSILON = 1.192092896 × 10-7 (odpowiednik Float32)
DBL_EPSILON = 2.2204460492503131 × 10-16 (odpowiednik Float64)
```

Zauważamy, że nasze wyniki wyszły bardzo podobne. (Nie ma odpowiednika Float16)

1.2 Liczba macheps a precyzja arytmetyki

Precyzję arytmetyki nazywamy liczbę opisaną wzorem:

$$\varepsilon = \frac{1}{2} \beta^{1-t}$$

- β to podstawa systemu. W naszym przypadku jest to 2, ponieważ w komputerach liczby zmiennoprzecinkowe reprezentowane są w systemie binarnym.
- t to liczba cyfr mantysy. Mantysa w formacie zmiennoprzecinkowym ma określoną liczbę bitów:

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

Precyzje arytmetyki wynoszą, więc odpowiednio:

Float16: $2^{-1} \cdot 2^{1-10} = 2^{-10}$
Float32: $2^{-1} \cdot 2^{1-23} = 2^{-23}$
Float64: $2^{-1} \cdot 2^{1-52} = 2^{-52}$

Gdy sprawdzimy w Julii wartości policzonych przed chwilą precyzji, z dokładnością do określonych typów, otrzymamy:

Float16: 0.000977
Float32: 1.1920929e-7
Float64: 2.220446049250313e-16

Powyższe wartości precyzji pokrywają się z wcześniej wyznaczonymi epsilonami maszynowymi, zatem nasuwa się prosty wniosek, że dla danej arytmetyki macheps jest równy jej precyzji.

1.3 Liczba maszynowa eta

Drugim podpunktem zadania było iteracyjne wyznaczenie liczby maszynowej eta. Jest to pierwsza liczba maszynowa większa od 0.0. Oto output programu, porównujący wyznaczone doświadczalnie liczby eta do wartości zwracanych przez funkcje `nextfloat(Float16(0.0))`, `nextfloat(Float32(0.0))`, `nextfloat(Float64(0.0))`.

Float16
iteracyjnie: 6.0e-8
funkcja nextfloat: 6.0e-8

Float32

iteracyjnie: 1.0e-45
funkcja nextfloat: 1.0e-45

Float64
iteracyjnie: 5.0e-324
funkcja nextfloat: 5.0e-324

Znów wyniki doświadczenia są zgodne z wartościami zwracanymi przez dedykowaną do tego funkcję.

1.4 Liczba eta a MIN_{sub}

Liczba MIN_{sub} jest najmniejszą liczbą, która jest nieznormalizowana, ponieważ przedstawienie jej w postaci znormalizowanej wymagałoby użycia wykładnika mniejszego niż dopuszczalny. Wzór na nią przedstawiony jest poniżej.

$$MIN_{sub} = 2^{1-t} \cdot 2^{c_{min}}$$

Gdzie:

- t to klasycznie liczba cyfr mantysy,
- c_{min} to minimalna możliwa do zapisania cecha,

$$c_{min} = -2^{d-1} + 2$$

- d liczba bitów przeznaczona na zapis cechy.

Dla badanych przez nas typów powyższe wartości to:

Float16: $c_{min} = -14$, $MIN_{sub} = 2^{-24}$
Float32: $c_{min} = -126$, $MIN_{sub} = 2^{-149}$
Float64: $c_{min} = -1022$, $MIN_{sub} = 2^{-1074}$

Po sprawdzeniu tych wartości w Julii:

Float16: 6.0e-8
Float32: 1.0e-45
Float64: 5.0e-324

otrzymujemy wynik, którego mogliśmy się spodziewać. Mianowicie MIN_{sub} jest równe co do wartości liczbie eta dla danej arytmetyki.

1.5 Liczba MIN_{nor}

Liczba MIN_{nor} to najmniejsza liczba w postaci znormalizowanej, którą da się zapisać. Wyliczyć ją można ze wzoru:

$$MIN_{nor} = 2^{c_{min}}$$

Mamy zatem:

Float16: $MIN_{nor} = 2^{-14}$
Float32: $MIN_{nor} = 2^{-126}$
Float64: $MIN_{nor} = 2^{-1022}$

Program liczący wartości `floatmin()` oraz MIN_{nor} dla danych typów zwraca nam takie wyniki:

```
-----Badanie floatmin-----  
Float16: 6.104e-5  
Float32: 1.1754944e-38  
Float64: 2.2250738585072014e-308  
-----Badanie MIN_nor-----  
Float16: 6.104e-5  
Float32: 1.1754944e-38  
Float64: 2.2250738585072014e-308
```

Wartości zwracane przez `floatmin()` pokrywają się z odpowiadającym mu MIN_{nor} w danej arytmetyce.

1.6 Liczba MAX

MAX to największa możliwa do wyrażenia liczba dla danego typu. Mantysa takiej liczby składa się z samych jedynek, a cecha osiąga najwyższą możliwą wartość.

Aby wyznaczyć taką liczbę potrzebujemy poprzednią liczbę maszynową jedynki. Ma ona mantysę wypełnioną jedynie jedynekami. Mnożenie tej liczby razy 2 zwiększa nam wykładnik, lecz mantysa pozostaje taka sama. Wobec tego otrzymujemy coraz to większą liczbę, za każdym razem mającą największą możliwą mantysę. Gdy osiągniemy infinity przerwyamy eksperyment, ponieważ poprzednia znaleziona liczba jest tą szukaną. Wynikiem naszego doświadczenia jest:

```
Float16  
iteracyjnie: 6.55e4  
funkcja floatmax: 6.55e4
```

```
Float32  
iteracyjnie: 3.4028235e38
```

funkcja floatmax: 3.4028235e38

Float64

iteracyjnie: 1.7976931348623157e308

funkcja floatmax: 1.7976931348623157e308

I znów zgodnie z oczekiwaniami wartości funkcji systemowej i doświadczenia się pokrywają. Gdy porównamy je z wartościami z pliku `float.h`, podobnie jak w przypadku macheps, otrzymamy wartości zbliżone a nie identyczne do tych wyznaczonych przez nas.

$FLT_MAX = 3.402823466 \times 10^{38}$ (odpowiednik Float32)

$DBL_MAX = 1.7976931348623158 \times 10^{308}$ (odpowiednik Float64)

1.7 Wnioski

Wszystkie obliczone powyżej wartości tzn. epsilon maszynowy, liczba eta, MIN_{sub} , MIN_{nor} , MAX zostały poprawnie wyznaczone i są zgodne z wartościami zwracanymi przez funkcje systemowe. Standard IEEE-754, mimo swojej powszechności, ma ograniczenia w dokładnym wyrażaniu liczb rzeczywistych. Jednak im więcej bitów zostaje przeznaczone na zapis liczby tym większa dokładność obliczeń, mniejsze wartości najmniejszych liczb dodatnich oraz odstęp między kolejnymi liczbami. Jednocześnie znacząco zwiększa się nam zakresmożliwych do reprezentacji wartości.

2 Zadanie 2

Celem zadania 2 jest wyznaczenie epsilon maszynowego za pomocą obliczenia wartości wyrażenia $3\left(\frac{4}{3} - 1\right) - 1$ i sprawdzenie czy sposób ten jest poprawny.

Wynik programu:

Float16

kahan_eps: -0.000977

eps: 0.000977

Float32

kahan_eps: 1.1920929e-7

eps: 1.1920929e-7

Float64

kahan_eps: -2.220446049250313e-16

eps: 2.220446049250313e-16

Wnioski

Wynik jest poprawny co do wartości bezwzględnej.

3 Zadanie 3

Zadanie polega na eksperymentalnym sprawdzeniu, że w arytmetyce Float64 liczby zmiennopozycyjne są równomiernie rozmieszczone w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$.

Sprawdzenie tego iteracyjnie dla wszystkich liczb należących do $[1, 2]$ byłoby bardzo kosztowne. Rozwiązaniem tego problemu jest zweryfikowanie wartości jedynie wokół końców. Skoro wartości po dodaniu delty będą ciągle zgodne z tymi rzeczywistymi, zarówno kiedy bierzemy początek i koniec przedziału, to na pewno nie nasąpiła zmiana wielkości delty wewnątrz przedziału.

Dla przedziału $[1, 2]$ wynik programu to:

Liczby w przedziale $[1, 2]$ są równomiernie rozmieszczone.

Teraz mamy zbadać jak rozmieszczone są liczby w przedziale $[1/2, 1]$ oraz $[2, 4]$
W arytmetyce Float64 naszą δ będzie:

$$\delta = 2^{\text{cecha}-1023} \cdot 2^{-52}$$

Jeśli chodzi o przedział $[1/2, 1]$ wynik naszego programu to:

Liczby w przedziale $[1/2, 1]$ są równomiernie rozmieszczone
`delta = 1.1102230246251565e-16 = 2^-53`

A dla przedziału $[2, 4]$:

Liczby w przedziale $[2, 4]$ są równomiernie rozmieszczone
`delta = 4.440892098500626e-16 = 2^-51`

Wnioski

W przedziale $[1/2, 1]$ δ wynosiła 2^{-53} . Dla przedziału $[1, 2]$ było to już 2^{-52} , a dla $[2, 4]$ - 2^{-51} . Nietrudno zauważyć, że krańce tych przedziałów to kolejne potęgi dwójki. Tak więc liczby zawierające się pomiędzy potęgami 2 są równomiernie rozmieszczone. Co więcej, ponieważ mantysa ma ograniczoną liczbę bitów, na której można ją zapisać każdy taki przedział zawiera tyle samo liczb. Dlatego w kolejnych przedziałach δ jest za każdym razem dwukrotnie większa.

4 Zadanie 4

W tym zadaniu należało eksperymentalnie wyznaczyć najmniejszą taką liczbę zmiennoprzecinkową z przedziału (1.0, 2.0), która spełnia warunek:

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

Output programu:

1.0000000572289969

Wnioski

Zgodnie z teorią matematyczną dla żadnej liczby x ten warunek nie powinien być spełniony. Program jednak znalazł taką liczbę, co wynika z ograniczonej precyzji mantysy we Float64, która ma 52 bity, co powoduje zaokrąglenia.

5 Zadanie 5

Zadanie polegało na wyliczeniu iloczynu skalarnego dwóch wektorów przy użyciu czterech różnych sposobów.

- technika "w przód" mnożymy odpowiadające sobie współrzędne i je dodajemy
- technika "w tył" mnożymy odpowiadające sobie współrzędne w kolejności od końca i je dodajemy
- od największego do najmniejszego (dodatnie liczby w porządku od największego do najmniejszego - sumujemy, ujemne liczby w porządku od najmniejszego do największego - sumujemy, następnie sumujemy dwie obliczone sumy)
- od najmniejszego do największego (przeciwnie do metody (c))

Wynik naszego programu:

Float32

real: -1.00657107000000e-11

forward: -0.4999443

backward: -0.4543457

biggest_to_smallest: -0.5

smallest_to_biggest: -0.5

Float64

real: -1.00657107000000e-11

forward: 1.0251881368296672e-10

```
backward: -1.5643308870494366e-10
biggest_to_smallest: 0.0
smallest_to_biggest: 0.0
```

Wnioski

Niestety żadna z powyższych metod nie dała nam precyzyjnego wyniku. Float64 gwarantuje trochę większą precyzję niż Float32, jednak nietrywialne znaczenie ma również kolejność wykonywania obliczeń.

6 Zadanie 6

Analiza wartości funkcji:

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

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

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

Sprawdź $x = 8^k$, dla $k = 1, 2, \dots, 20$

x	f(x)	g(x)
k = 1	0.0077822185373186414	0.0077822185373187065
k = 2	0.00012206286282867573	0.00012206286282875901
k = 3	1.9073468138230965e-6	1.907346813826566e-6
k = 4	2.9802321943606103e-8	2.9802321943606116e-8
k = 5	4.656612873077393e-10	4.6566128719931904e-10
k = 6	7.275957614183426e-12	7.275957614156956e-12
k = 7	1.1368683772161603e-13	1.1368683772160957e-13
k = 8	1.7763568394002505e-15	1.7763568394002489e-15
k = 9	0.0	2.7755575615628914e-17
k = 10	0.0	4.336808689942018e-19
k = 11	0.0	6.776263578034403e-21
k = 12	0.0	1.0587911840678754e-22
k = 13	0.0	1.6543612251060553e-24
k = 14	0.0	2.5849394142282115e-26
k = 15	0.0	4.0389678347315804e-28
k = 16	0.0	6.310887241768095e-30
k = 17	0.0	9.860761315262648e-32
k = 18	0.0	1.5407439555097887e-33
k = 19	0.0	2.407412430484045e-35
k = 20	0.0	3.76158192263132e-37

Wnioski

W sensie matematycznym funkcje są równe, to znaczy powinniśmy otrzymywać takie same wyniki. Jednak ze względu na ograniczoną precyzję dla bardzo małych x $\sqrt{x^2 + 1} \approx 1$. Na ćwiczeniach z obliczeń naukowych rozwiązywaliśmy zadanie, w którym pokazaliśmy, że odjemowanie liczb o bardzo zbliżonej wartości powoduje duży spadek precyzji wyniku. Jest to spowodowane utratą liczb znaczących. Wobec tego w funkcji $f(x)$, gdy odejmujemy 1 od $\sqrt{x^2 + 1}$ tracimy dokładność i otrzymujemy zera (od pewnego momentu). Funkcja $g(x)$ jest w tym przypadku bardziej dokładna.