Zofia Tarchalska - 279699

Wprowadznie

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 epsilona i sprawdzanie czy dodanie go do jedynki będzie skutkowało zwiększeniem wyniku. Oto output programu, porównujący wyznaczone doświadczalnie epsilony do wartości zwracanych przez funckje `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` $= 1.192092896 \times 10^{-7}$ (odpowiednik Float32)
`DBL_EPSILON` $= 2.2204460492503131 \times 10^{-16}$ (odpowiednik Float64)

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

1.2 Liczba macheps a precyzja arytmetyki

Precyzją 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 spradzimy 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 funckję.

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 jedynkami. 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 przerywamy 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 wyzanczonych 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 odstępy między kolejnymi liczbami. Jednocześnie znacząco zwiększa się nam zakres możliwych do reprezentacji wartości.

2 Zadanie 2

Celem zadania 2 jest wyznaczenie epsilona 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 kojenych 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 precyzyjengo wyniku. Float64 gwarantuje trochę większą precyzję niż Float32, jednak nietrywialne znaczenie ma również kolejność wykonywania obliczeń.

6 Zadanie 6

Analiza wartości funckji:

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

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

Dla x = 8^{-1}, 8^{-2}, 8^{-3}, ...
Sprawdzę x = 8^k, dla k = 1, 2, ..., 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 funckje 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 cyfr znaczących. Wobec tego w funckji $f(x)$, gdy odejmujemy 1 od $\sqrt{x^2+1}$ tracimy dokładność i otrzymujemy zera (od pewnego momentu). Funckja $g(x)$ jest w tym przypadku bardziej dokładna.

7 Zadanie 7

Pochodną funkcji możemy przybliżyć używając poniższego wzoru:

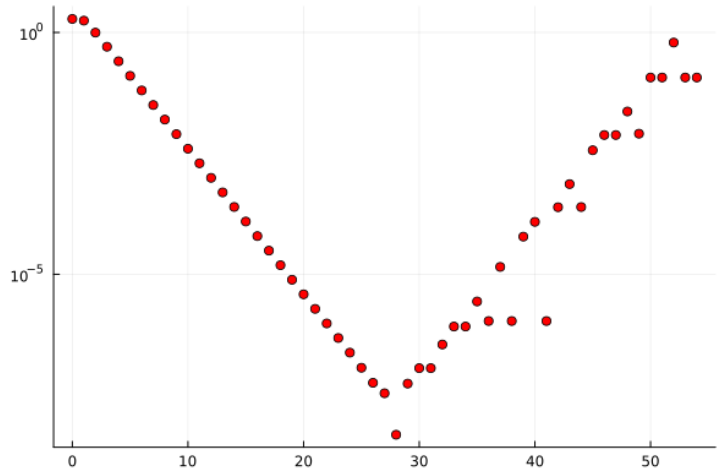
$$f'(x_0) \approx \tilde{f}'(x_0) = \frac{f(x_0+h) - f(x_0)}{h}.$$

Zadanie polega na skorzystaniu z tego wzoru i obliczeniu przybliżonej wartości pochodnej funkcji w punkcie $x_0 = 1$ oraz dla $h = 2^{-n}$ gdzie n = 1, 2, ..., 54.

n	f_tilde	|f_tilde - f'|
1	1.8704413979316472	1.753499116243109
2	1.1077870952342974	0.9908448135457593
3	0.6232412792975817	0.5062989976090435
4	0.3704000662035192	0.253457784514981
5	0.24344307439754687	0.1265007927090087
6	0.18009756330732785	0.0631552816187897
7	0.1484913953710958	0.03154911368255764
8	0.1327091142805159	0.015766832591977753
9	0.1248236929407085	0.007881411252170345
10	0.12088247681106168	0.0039401951225235265
11	0.11891225046883847	0.001969968780300313
12	0.11792723373901026	0.000984949520504721099
13	0.11743474961076572	0.0004924679222275685
14	0.11718851362093119	0.0002462319323930373
15	0.11706539714577957	0.00012311545724141837
16	0.11700383928837255	6.155759983439424e-5
17	0.11697306045971345	3.077877117529937e-5
18	0.11695767106721178	1.5389378673624776e-5
19	0.11694997636368498	7.694675146829866e-6
20	0.11694612901192158	3.8473233834324105e-6
21	0.1169442052487284	1.9235601902423127e-6
22	0.11694324295967817	9.612711400208696e-7
23	0.11694276239722967	4.807086915192826e-7

24	0.11694252118468285	2.394961446938737e-7
25	0.116942398250103	1.1656156484463054e-7
26	0.11694233864545822	5.6956920069239914e-8
27	0.11694231629371643	3.460517827846843e-8
28	0.11694228649139404	4.802855890773117e-9
29	0.11694222688674927	5.480178888461751e-8
30	0.11694216728210449	1.1440643366000813e-7
31	0.11694216728210449	1.1440643366000813e-7
32	0.11694192886352539	3.5282501276157063e-7
33	0.11694145202636719	8.296621709646956e-7
34	0.11694145202636719	8.296621709646956e-7
35	0.11693954467773438	2.7370108037771956e-6
36	0.116943359375	1.0776864618478044e-6
37	0.1169281005859375	1.4181102600652196e-5
38	0.116943359375	1.0776864618478044e-6
39	0.11688232421875	5.9957469788152196e-5
40	0.1168212890625	0.0001209926260381522
41	0.116943359375	1.0776864618478044e-6
42	0.11669921875	0.0002430629385381522
43	0.1162109375	0.0007313441885381522
44	0.1171875	0.0002452183114618478
45	0.11328125	0.003661031688538152
46	0.109375	0.007567281688538152
47	0.109375	0.007567281688538152
48	0.09375	0.023192281688538152
49	0.125	0.008057718311461848
50	0.0	0.11694228168853815
51	0.0	0.11694228168853815
52	-0.5	0.6169422816885382
53	0.0	0.11694228168853815
54	0.0	0.11694228168853815

Wyniki działania programu prezentują się powyżej. Dla zwiększenia czytelności poniżej dodany jest wykres prezentujący wartości błędów.



Możemy zauważyć, że najmniejszy błąd jest dla n = 28.
Teraz przyjrzyjmy się wartościom przyjmowanym dla 1 + h

n	1+h	f(1+h)	f(1+h)-f(1)
1	1.5	0.7866991871732747	0.9352206989658236
2	1.25	0.12842526201602544	0.27694677380857435
3	1.125	-0.0706163518803512	0.07790515991219771
4	1.0625	-0.12537150765482896	0.02315000413771995
5	1.03125	-0.14091391571762557	0.0076075960749233396
6	1.015625	-0.1457074873658719	0.002814024426676997 6
7	1.0078125	-0.14736142276621222	0.0011600890263366859
8	1.00390625	-0.14800311681489065	0.0005183949776582653
9	1.001953125	-0.1482777155172741	0.00024379627527482128
10	1.0009765625	-0.1484034624987881	0.00011804929376080242
11	1.00048828125	-0.14846344917024967	5.806262229923753e-5
12	1.000244140625	-0.14849272096399935	2.8790828549563052e-5
13	1.0001220703125	-0.14850717649596556	1.4335296583345425e-5
14	1.00006103515625	-0.14851435917330935	7.152619239558788e-6
15	1.000030517578125	-0.14851793924014578	3.57255240313048e-6
16	1.0000152587890625	-0.1485197264556457	1.7853369032039268e-6
17	1.0000076293945312	-0.14852061935892114	8.924336277749134e-7
18	1.0000038146972656	-0.1485210656344409	4.4615810801396094e-7
19	1.0000019073486328	-0.14852128872817139	2.2306437752472874e-7
20	1.0000009536743164	-0.14852140026402927	1.1152851964180144e-7
21	1.0000004768371582	-0.1485214560292064	5.576334249912662e-8
22	1.000000238418579	-0.1485214839111071	2.7881441821975272e-8
23	1.0000001192092896	-0.1485214978518853	1.3940663623479566e-8
24	1.0000000596046448	-0.14852150482223148	6.970317434351614e-9
25	1.0000000298023224	-0.14852150830739386	3.4851550534398257e-9
26	1.0000000149011612	-0.14852151004997227	1.7425766385414931e-9

```
27   1.0000000074505806      -0.14852151092126076     8.712881527372929e-10
28   1.0000000037252903      -0.14852151135690494     4.35643965346344e-10
29   1.0000000018626451      -0.14852151157472704     2.1782187165086953e-10
30   1.0000000009313226      -0.14852151168363803     1.0891088031428353e-10
31   1.0000000004656613      -0.14852151173809347     5.4455440157141766e-11
32   1.0000000002328306      -0.14852151176532125     2.722766456741965e-11
33   1.0000000001164153      -0.14852151177893513     1.3613776772558595e-11
34   1.0000000000582077      -0.14852151178574202     6.806888386279297e-12
35   1.0000000000291038      -0.14852151178914552     3.403388681988417e-12
36   1.000000000014552       -0.14852151179084716     1.70174985214544e-12
37   1.000000000007276       -0.14852151179169815     8.507639037702575e-13
38   1.000000000003638       -0.14852151179212347     4.2543746303636e-13
39   1.000000000001819       -0.1485215117923363      2.1260770921571748e-13
40   1.0000000000009095      -0.14852151179244266     1.0624834345662748e-13
41   1.0000000000004547      -0.14852151179249573     5.3179682879545e-14
42   1.0000000000002274      -0.14852151179252238     2.653433028854124e-14
43   1.0000000000001137      -0.1485215117925357      1.3211653993039363e-14
44   1.0000000000000568      -0.14852151179254225     6.661338147750939e-15
45   1.0000000000000284      -0.1485215117925457      3.219646771412954e-15
46   1.0000000000000142      -0.14852151179254736     1.5543122344752192e-15
47   1.000000000000007       -0.14852151179254813     7.771561172376096e-16
48   1.0000000000000036      -0.14852151179254858     3.3306690738754696e-16
49   1.0000000000000018      -0.1485215117925487      2.220446049250313e-16
50   1.0000000000000009      -0.1485215117925489      0.0
51   1.0000000000000004      -0.1485215117925489      0.0
52   1.0000000000000002      -0.14852151179254902     -1.1102230246251565e-16
53   1.0                     -0.1485215117925489      0.0
54   1.0                     -0.1485215117925489      0.0
```

Wartość funckji w punkcie 1.0 wyonsi natomiast:
`f(1.0) = -0.1485215117925489`
We wzorze na przybliżoną pochodną w liczniku pojawia się odejmowanie. I tak samo jak dla poprzedniego zadnia odejmowanie podobnych wartości od siebie (gdy h jest małe) skutkuje utratą cyfr zanczących a w efekcie niską dokładnośc wyniku. Dla h, które są mniejsze niż 2^{-53} w arytmetyce Float64 mamy 1 + h = 1. Dlatego otrzymujemy wyniki równe 0.

Wnioski

Nie wszystkie wzory da się łatwo przekształcić w taki sposób, aby uniknąć utraty cyfr znaczących przy odejmowaniu. Występujące błędy numeryczne pokazują, że w praktyce matematyczna intuicja - zgodnie z którą podstawianie coraz mniejszych wartości powinno zwiększać dokładność przybliżenia - nie zawsze się sprawdza.