Sprawozdanie

Paweł Krzyszczak

Październik 2024

1 Zadanie 1

1.1 Wyznaczanie epsilonu maszynowego (macheps)

Celem było wyznaczenie iteracyjnie epsilonu maszynowego (macheps) dla typów `Float16`, `Float32` oraz `Float64`, a następnie porównanie wyników z wartościami zwracanymi przez funkcję `eps` oraz z wartościami zawartymi w pliku nagłówkowym `float.h`.

Epsilon maszynowy (macheps) został wyznaczony iteracyjnie poprzez dzielenie wartości początkowej przez dwa aż do osiągnięcia najmniejszej liczby `macheps > 0`, dla której $1.0 + \texttt{macheps} > 1.0$. Dla każdego typu `T` (`Float16`, `Float32`, `Float64`)

```
>julia zad1.1.jl
Epsilon maszynowy dla Float16:
Wbudowany: 0.000977
Moj:  0.000977
Epsilon maszynowy dla Float32:
Wbudowany: 1.1920929e-7
Moj:  1.1920929e-7
Epsilon maszynowy dla Float64:
Wbudowany: 2.220446049250313e-16
Moj:  2.220446049250313e-16
```

Wartości uzyskane iteracyjnie były zgodne z wynikami funkcji `eps(Float16)`, `eps(Float32)`, `eps(Float64)`, co potwierdza, że metoda iteracyjna poprawnie wyznacza macheps dla różnych typów zmiennopozycyjnych, zgodnie ze standardem IEEE 754.

1.2 Wyznaczanie minimalnej dodatniej liczby (eta)

Celem było wyznaczenie iteracyjnie minimalnej dodatniej liczby eta > 0.0, którą można zapisać w arytmetyce zmiennopozycyjnej dla `Float16`, `Float32` i `Float64`, oraz porównanie wyników z wartościami zwracanymi przez `nextfloat(0.0)` dla każdego z typów.

Wartość eta obliczono poprzez iteracyjne dzielenie liczby 1 przez 2 aż do uzyskania minimalnej dodatniej liczby, która jest większa od 0.

```
>julia zad1.2.jl
Eta dla Float16:
Wbudowany: 6.0e-8
Moj:  6.0e-8
Eta dla Float32:
Wbudowany: 1.0e-45
Moj:  1.0e-45
Eta dla Float64:
Wbudowany: 5.0e-324
Moj:  5.0e-324
```

Uzyskane wartości eta były zgodne z wynikami `nextfloat(Float16(0.0))`, `nextfloat(Float32(0.0))`, i `nextfloat(Float64(0.0))`, co potwierdza zgodność wyników z modelem standardu IEEE 754 oraz poprawność metody iteracyjnej.

1.3 Wyznaczanie maksymalnej liczby (MAX)

Celem było wyznaczenie największej liczby zmiennopozycyjnej (MAX), którą można przedstawić dla `Float16`, `Float32` oraz `Float64`, zanim osiągnie się nieskończoność (`Inf`), oraz porównanie wyników z wartościami zwracanymi przez `floatmax`.

Wartość `MAX` wyznaczono poprzez iteracyjne mnożenie początkowej liczby 1.0 przez 2, aż uzyskana wartość przekroczy zakres typu `T`, co prowadzi do osiągnięcia nieskończoności.

```
>julia zad1.3.jlMaximum dla Float16:
Wbudowany: 6.55e4
Moj:  6.55e4
Maximum dla Float32:
Wbudowany: 3.4028235e38
Moj:  3.4028235e38
Maximum dla Float16:
Wbudowany: 6.55e4
Moj:  6.55e4
```

Uzyskane wartości `MAX` były zgodne z wynikami funkcji `floatmax(Float16)`, `floatmax(Float32)`, i `floatmax(Float64)`, a także z danymi zawartymi w `float.h`. Potwierdza to poprawność wyznaczenia maksymalnej liczby dla każdego typu i zgodność wyników z arytmetyką IEEE 754.

2 Zadanie 2

Celem zadania było sprawdzenie eksperymentalnie w języku Julia słuszności stwierdzenia Kahna, że epsilon maszynowy (macheps) można uzyskać za pomocą wyrażenia $3\left(\frac{4}{3}-1\right)-1$ w arytmetyce zmiennopozycyjnej. Eksperyment miał zostać przeprowadzony dla wszystkich typów zmiennopozycyjnych w standardzie IEEE 754: `Float16`, `Float32` oraz `Float64`.

Epsilon maszynowy (macheps) jest najmniejszą liczbą $\varepsilon > 0$, dla której $1 + \varepsilon > 1$ w danej arytmetyce zmiennopozycyjnej, co stanowi miarę precyzji tej arytmetyki. Kahan zauważył, że wyrażenie $3\left(\frac{4}{3}-1\right)-1$ może przybliżyć tę wartość ze względu na zaokrąglenia w obliczeniach zmiennopozycyjnych.

Zadanie zostało zaimplementowane w języku Julia. Dla każdego z typów `Float16`, `Float32` oraz `Float64` obliczono wartość wyrażenia Kahna oraz porównano ją z wartością zwracaną przez funkcję `eps`, która zwraca epsilon maszynowy dla danego typu.

```
>julia zad2.jl
Epsilon maszynowy dla Float16:
Wbudowany: 0.000977
Kahans:  -0.000977
Epsilon maszynowy dla Float32:
Wbudowany: 1.1920929e-7
Kahans:  1.1920929e-7
Epsilon maszynowy dla Float64:
Wbudowany: 2.220446049250313e-16
Kahans:  -2.220446049250313e-16
```

Eksperyment potwierdził, że metoda Kahna jest efektywnym sposobem przybliżenia epsilonu maszynowego dla typów zmiennopozycyjnych `Float16`, `Float32`, i `Float64`. Uzyskane wartości są zgodne z epsilonem maszynowym zwracanym przez funkcję `eps` w języku Julia, co świadczy o wysokiej precyzji metody Kahna przy wyznaczaniu macheps.

3 Zadanie 3

Celem zadania jest eksperymentalne sprawdzenie rozmieszczenia liczb zmiennoprzecinkowych typu `Float64` w standardzie IEEE 754 w przedziałach $[1, 2]$, $\left[\frac{1}{2}, 1\right]$ i $[2, 4]$ w języku Julia.

Liczby zmiennoprzecinkowe `Float64` w standardzie IEEE 754 są reprezentowane jako:

$$x = 1 + k\delta$$

gdzie $k = 1, 2, \ldots, 2^{52} - 1$ oraz $\delta = 2^{-52}$. Oznacza to, że liczby są równomiernie rozmieszczone w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$.

W celu sprawdzenia rozmieszczenia tych liczb w innych przedziałach należy uwzględnić właściwości reprezentacji zmiennoprzecinkowej, co pozwala odpowiednio skalować kroki między liczbami w różnych zakresach.

Eksperyment przeprowadzono w języku Julia. Użyto funkcji `bitstring` do obserwacji różnic w reprezentacji binarnej liczb zmiennoprzecinkowych.

```
>julia zad3.jl
Liczby zmiennopozycyjne w przedziale [1, 2]:
bitstring: 0011111111110000000000000000000000000000000000000000000000000000
bitstring: 0011111111110000000000000000000000000000000000000000000000000001
bitstring: 0011111111110000000000000000000000000000000000000000000000000010
bitstring: 0011111111110000000000000000000000000000000000000000000000000011
...
bitstring: 0011111111111111111111111111111111111111111111111111111111111101
bitstring: 0011111111111111111111111111111111111111111111111111111111111110
bitstring: 0011111111111111111111111111111111111111111111111111111111111111
bitstring: 0100000000000000000000000000000000000000000000000000000000000000

Liczby zmiennopozycyjne w przedziale [1/2, 1]:
bitstring: 0011111111100000000000000000000000000000000000000000000000000000
bitstring: 0011111111100000000000000000000000000000000000000000000000000010
bitstring: 0011111111100000000000000000000000000000000000000000000000000100
bitstring: 0011111111100000000000000000000000000000000000000000000000000110
...
bitstring: 0011111111101111111111111111111111111111111111111111111111111010
bitstring: 0011111111101111111111111111111111111111111111111111111111111100
bitstring: 0011111111101111111111111111111111111111111111111111111111111110
bitstring: 0011111111110000000000000000000000000000000000000000000000000000

Liczby zmiennopozycyjne w przedziale [2, 4]:
bitstring: 0100000000000000000000000000000000000000000000000000000000000000
bitstring: 0100000000000000000000000000000000000000000000000000000000000000
bitstring: 0100000000000000000000000000000000000000000000000000000000000001
bitstring: 0100000000000000000000000000000000000000000000000000000000000010
...
bitstring: 0100000000001111111111111111111111111111111111111111111111111110
bitstring: 0100000000001111111111111111111111111111111111111111111111111111
bitstring: 0100000000010000000000000000000000000000000000000000000000000000
bitstring: 0100000000010000000000000000000000000000000000000000000000000000
```

Przeprowadzone eksperymenty wykazały, że liczby zmiennoprzecinkowe `Float64` w przedziale $[1, 2]$ są rozmieszczone równomiernie z krokiem $\delta = 2^{-52}$, co jest zgodne z teoretycznym założeniem.

Dla przedziału $\left[\frac{1}{2}, 1\right]$, liczby zmiennoprzecinkowe są również równomiernie rozmieszczone, ale krok między kolejnymi wartościami jest równy $\delta = 2^{-53}$ — co wynika z niższej wartości potęgi w tym przedziale.

Natomiast w przedziale $[2, 4]$ liczby są rozmieszczone co $2\delta = 2^{-51}$, czyli dwa razy większym krokiem niż w przedziale $[1, 2]$.

Eksperyment potwierdził, że liczby zmiennoprzecinkowe `Float64` w standardzie IEEE 754 są równomiernie rozmieszczone w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$. W innych przedziałach, takich jak $\left[\frac{1}{2}, 1\right]$ i $[2, 4]$, kroki ulegają odpowiednim zmianom: są mniejsze w przedziale $\left[\frac{1}{2}, 1\right]$ oraz większe w przedziale $[2, 4]$.

4 Zadanie 4

Liczby zmiennopozycyjne w arytmetyce komputerowej reprezentowane są zgodnie z ustalonym standardem IEEE 754, który w przypadku typu `Float64` pozwala na przechowywanie liczb zmiennopozycyjnych z podwójną precyzją. W praktyce oznacza to, że liczby te mają ograniczoną precyzję, co prowadzi do błędów zaokrągleń.

W zadaniu celem jest znalezienie liczby x z zakresu $(1, 2)$, dla której wyniki operacji arytmetycznych $x \cdot (1/x)$ nie są równe 1, co wynika z błędów zaokrągleń.

Został napisany program w języku Julia, który przeszukuje przedział $(1, 2)$ i znajduje liczbę x, dla której $x \cdot (1/x) \neq 1$. Program iteruje przez liczby z określonym krokiem, weryfikując warunek zadania.

```
>julia zad4.jl
Znalezione x:1.000000057228997
Sprawdzenie: x * (1/x) =0.9999999999999999
W zapisie bitowym: 0011111111110000000000000000000000001111010111001011111100101010
```

W eksperymencie udało się znaleźć najmniejszą liczbę x, dla której $x \cdot (1/x) \neq 1$, co potwierdza, że błędy zaokrągleń wpływają na operacje w arytmetyce zmiennopozycyjnej.

5 Zadanie 5

Celem było przeprowadzenie eksperymentu, w którym iloczyn skalarny tych wektorów obliczano na cztery różne sposoby, przy użyciu zarówno pojedynczej (Float32), jak i podwójnej precyzji (Float64), oraz porównanie wyników z wartością referencyjną $-1.00657107000000 \times 10^{-11}$.

```
>julia zad5.jl
                        Float32         Float64
In order:               -0.4999443      1.0251881368296672e-10
In reversed order:      -0.4543457      -1.5643308870494366e-10
In descending order:    -0.5            0.0
In ascending order:     -0.5            0.0
```

Żadna z prób obliczenia tego iloczynu nie jest dostatecznie bliska do faktycznego wyniku aby uznać ją za udaną.

6 Zadanie 6

Celem tego zadania jest obliczenie wartości funkcji:

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

oraz

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

dla malejących wartości argumentu $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$ przy użyciu arytmetyki Float64 w języku Julia. Mimo że $f(x) = g(x)$ w teorii, w obliczeniach komputerowych wyniki mogą się różnić. Zadanie polega na analizie otrzymanych wyników oraz na ocenie, które z nich są wiarygodne.

Funkcje $f(x)$ i $g(x)$ są teoretycznie równe. Jednak w obliczeniach komputerowych mogą wystąpić różnice wynikające z ograniczeń dokładności arytmetyki zmiennoprzecinkowej (Float64). Przy małych wartościach x, funkcja $f(x)$ jest podatna na utratę cyfr znaczących, co może prowadzić do nieprecyzyjnych wyników. Funkcja $g(x)$, dzięki wyrażeniu unikania bezpośredniego odejmowania liczby bliskiej jedności od liczby o podobnej wartości, powinna być bardziej stabilna.

Obliczenia wykonano dla kolejnych wartości $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$. Wyniki zrealizowano w języku Julia przy użyciu arytmetyki Float64.

```
>julia zad6.jl
x = 8^-1,  f(x) = 0.0077822185373186414, g(x) = 0.0077822185373187065
x = 8^-2,  f(x) = 0.00012206286282867573, g(x) = 0.00012206286282875901
x = 8^-3,  f(x) = 1.9073468138230965e-6, g(x) = 1.907346813826566e-6
x = 8^-4,  f(x) = 2.9802321943606103e-8, g(x) = 2.9802321943606116e-8
x = 8^-5,  f(x) = 4.656612873077393e-10, g(x) = 4.6566128719931904e-10
x = 8^-6,  f(x) = 7.275957614183426e-12, g(x) = 7.275957614156956e-12
x = 8^-7,  f(x) = 1.1368683772161603e-13, g(x) = 1.1368683772160957e-13
x = 8^-8,  f(x) = 1.7763568394002505e-15, g(x) = 1.7763568394002489e-15
x = 8^-9,  f(x) = 0.0, g(x) = 2.7755575615628914e-17
x = 8^-10,  f(x) = 0.0, g(x) = 4.336808689942018e-19
...
x = 8^-166,  f(x) = 0.0, g(x) = 7.466108948025751e-301
x = 8^-167,  f(x) = 0.0, g(x) = 1.1665795231290236e-302
x = 8^-168,  f(x) = 0.0, g(x) = 1.8227805048890994e-304
x = 8^-169,  f(x) = 0.0, g(x) = 2.848094538889218e-306
x = 8^-170,  f(x) = 0.0, g(x) = 4.450147717014403e-308
x = 8^-171,  f(x) = 0.0, g(x) = 6.953355807835e-310
x = 8^-172,  f(x) = 0.0, g(x) = 1.086461844974e-311
x = 8^-173,  f(x) = 0.0, g(x) = 1.69759663277e-313
x = 8^-174,  f(x) = 0.0, g(x) = 2.65249474e-315
x = 8^-175,  f(x) = 0.0, g(x) = 4.144523e-317
x = 8^-176,  f(x) = 0.0, g(x) = 6.4758e-319
```

```
x = 8^-177,  f(x) = 0.0, g(x) = 1.012e-320
x = 8^-178,  f(x) = 0.0, g(x) = 1.6e-322
x = 8^-179,  f(x) = 0.0, g(x) = 0.0
...
x = 8^-358,  f(x) = 0.0, g(x) = 0.0
```

Z analizy wyników wynika, że funkcja $g(x)$ wykazuje większą stabilność numeryczną przy małych wartościach x. W przypadku funkcji $f(x)$ występuje zjawisko utraty cyfr znaczących, co prowadzi do mniejszej dokładności obliczeń. W związku z tym, dla małych wartości x, bardziej wiarygodne wyniki daje funkcja $g(x)$.

Na podstawie przeprowadzonych obliczeń można stwierdzić, że mimo teoretycznej równości funkcji $f(x)$ i $g(x)$, w praktyce komputerowej różnice wyników wynikają z ograniczeń precyzji arytmetyki Float64. Funkcja $g(x)$, dzięki swojej formie, jest bardziej odporna na błędy numeryczne i dla małych wartości x jest bardziej wiarygodna. W przyszłości przy obliczeniach tego typu warto rozważać stosowanie alternatywnych formuł, które minimalizują ryzyko utraty cyfr znaczących.

7 Zadanie 7

Celem zadania było obliczenie przybliżonej wartości pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x_0 = 1$ za pomocą wzoru:

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

Następnie, dla różnych wartości $h = 2^{-n}$ (gdzie $n = 0, 1, 2, \ldots, 54$), obliczone zostały błędy w porównaniu z dokładną wartością pochodnej $f'(x_0)$.

Obliczyliśmy pochodną funkcji $f(x)$ analitycznie:

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

Podstawiając $x = 1$:

$$f'(1) = \cos(1) - 3\sin(3) \approx -1.660$$

Potęga dwójki exp	**Przybliżenie $f'(1)$**	**Błąd względny**
0	2.0179892252685967	1.9010469435800585
-1	1.8704413979316472	1.753499116243109
-2	1.1077870952342974	0.9908448135457594
...	...	...
-27	0.11694231629371643	3.4605178375612944e-8
-28	0.11694228649139404	4.802855987917631e-9
-29	0.11694222688674927	5.4801788787472994e-8
...	...	...
-52	-0.5	0.616942281688538
-53	0.0	0.11694228168853806
-54	0.0	0.11694228168853806

Eksperyment pokazał, że metoda różnicy skończonej daje zadowalające wyniki dla odpowiednich wartości kroku h. Zbyt duży krok prowadzi do dużych błędów przybliżenia, podczas gdy zbyt mały krok wprowadza błędy związane z precyzją reprezentacji liczby zmiennoprzecinkowej.