Obliczenia naukowe: Sprawozdanie 1

Szymon Hładyszewski

21 października 2025

1 Zadanie 1

1.1 Opis problemu

Zadanie polegało na ręcznym wyznaczeniu epsilona maszynowego, ety oraz liczby MAX dla typów Float16, Float32, Float64. Ponadto uzyskane wyniki należało porównać z funkcjami wbudowanymi w języku Julia oraz danymi zawartymi w pliku nagłówkowym `float.h` w języku C.

1.2 Rozwiązanie

Rozwiązanie tego problemu zostało zaimplementowane w pliku `zadanie1.jl`.

1.3 Wyniki i interpretacja

Typ	Epsilon maszynowy (ręcznie)	Epsilon maszynowy (Julia)	Epsilon maszynowy (`float.h`)
Float16	9.77×10^{-4}	9.77×10^{-4}	X
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}	1.192093×10^{-7}
Float64	$2.220446049250313 \times 10^{-16}$	$2.220446049250313 \times 10^{-16}$	2.220446×10^{-16}

Tabela 1: Porównanie epsilona maszynowego dla różnych typów zmiennoprzecinkowych

Typ	Eta (ręcznie)	Eta (Julia)
Float16	6×10^{-8}	6×10^{-8}
Float32	1×10^{-45}	1×10^{-45}
Float64	5×10^{-324}	5×10^{-324}

Tabela 2: Porównanie ety wyznaczanej ręcznie oraz tej domyślnej dla różnych typów zmiennoprzecinkowych

Typ	MAX (ręcznie)	MAX (Julia)	MAX (`float.h`)
Float16	6.55×10^{4}	6.55×10^{4}	X
Float32	3.4028235×10^{38}	3.4028235×10^{38}	3.402823×10^{38}
Float64	$1.7976931348623157 \times 10^{308}$	$1.7976931348623157 \times 10^{308}$	1.797693×10^{308}

Tabela 3: Porównanie MAX dla różnych typów zmiennoprzecinkowych wyznaczanym ręcznie, za pomocą funkcji wbudowanej oraz z biblioteki `float.h`

1.4 Wnioski

Wyżej przedstawione tabele pokazują, że wartości liczb `macheps`, eta i MAX wyznaczone ręcznie pokrywają się z funkcjami wbudowanymi w języku Julia oraz są podobne do tych z biblioteki `float.h` w języku

Typ	Wartość `floatmin()`	Wartość `MIN_nor`
Float32	$1.1754944 \times 10^{-38}$	1.2×10^{-38}
Float64	$2.2250738585072014 \times 10^{-308}$	2.2×10^{-308}

Tabela 4: Porównanie wartości minimalnych z funkcji `floatmin()` oraz `MIN_nor` podanej na wykładzie

C. Niewielkie różnice mogą wynikać z zaokrągleń lub specyficznych implementacji w różnych językach programowania.

2 Zadanie 2

2.1 Opis problemu

Zadanie polegało na eksperymentalnym sprawdzeniu, czy wyrażenie

$$3\left(\frac{4}{3}-1\right)-1$$

będące epsilonem maszynowym jest równe zero dla typów Float16, Float32, Float64.

2.2 Rozwiązanie

Rozwiązanie tego problemu zostało zaimplementowane w pliku `zadanie2.jl`.

2.3 Wyniki i interpretacja

Typ	Wynik eksperymentalny (ręcznie)	Wartość epsilon (Julia)
Float16	-9.77×10^{-4}	9.77×10^{-4}
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}
Float64	$-2.220446049250313 \times 10^{-16}$	$2.220446049250313 \times 10^{-16}$

Tabela 5: Porównanie wyniku eksperymentalnego (wzór Kahna) z wartością epsilon maszynowego dla różnych typów zmiennoprzecinkowych

2.4 Wnioski

Wyniki eksperymentalne pokazują, że wyrażenie

$$3\left(\frac{4}{3}-1\right)-1$$

nie jest równe zero dla żadnego z badanych typów. Uzyskane wyniki są takie same jak wartości epsilona maszynowego. Różnica wynika z braku zastosowania funkcji `abs()` na wynik eksperymentu oraz z różnej ostatniej cyfry mantysy w poszczególnych typach Float.

3 Zadanie 3

3.1 Opis problemu

Zadanie polegało na eksperymentalnym sprawdzeniu, że w arytmetyce Float64 liczby zmiennopozycyjne są równomiernie rozmieszczone na przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$. Ponadto należało zbadać wartość kroku dla przedziałów $[0.5, 1]$ oraz $[2, 4]$.

3.2 Rozwiązanie

Rozwiązanie tego problemu zostało zaimplementowane w pliku `zadanie3.jl`. Pierwsza funkcja służyła do wyznaczenia kroku δ na danym przedziale oraz sprawdzenia, czy liczby są równomiernie rozmieszczone. Druga funkcja służyła do wyświetlenia zapisu bitowego kolejnych liczb w tymże przedziale.

3.3 Wyniki i interpretacja

3.3.1 Przedział $[1, 2]$

Krok delta wynosi $\delta = 2^{-52} \approx 2.220446049250313 \times 10^{-16}$. Liczby są równomiernie rozmieszczone. Przykładowy zapis bitowy:

Iteracja	Zapis bitowy (znak, cecha, mantysa)
1	0 01111111111 0000000000000000000000000000000000000000000000000000
2	0 01111111111 0000000000000000000000000000000000000000000000000001
3	0 01111111111 0000000000000000000000000000000000000000000000000010
4	0 01111111111 0000000000000000000000000000000000000000000000000011
5	0 01111111111 0000000000000000000000000000000000000000000000000100

Tabela 6: Zapis bitowy kolejnych liczb na przedziale $[1, 2]$

3.3.2 Przedział $[0.5, 1]$

Krok delta wynosi $\delta = 2^{-53} \approx 1.1102230246251565 \times 10^{-16}$.

Iteracja	Zapis bitowy (znak, cecha, mantysa)
1	0 01111111110 0000000000000000000000000000000000000000000000000000
2	0 01111111110 0000000000000000000000000000000000000000000000000001
3	0 01111111110 0000000000000000000000000000000000000000000000000010
4	0 01111111110 0000000000000000000000000000000000000000000000000011
5	0 01111111110 0000000000000000000000000000000000000000000000000100

Tabela 7: Zapis bitowy kolejnych liczb na przedziale $[0.5, 1]$

3.3.3 Przedział $[2, 4]$

Krok delta wynosi $\delta = 2^{-51} \approx 4.440892098500626 \times 10^{-16}$.

Iteracja	Zapis bitowy (znak, cecha, mantysa)
1	0 10000000000 0000000000000000000000000000000000000000000000000000
2	0 10000000000 0000000000000000000000000000000000000000000000000001
3	0 10000000000 0000000000000000000000000000000000000000000000000010
4	0 10000000000 0000000000000000000000000000000000000000000000000011
5	0 10000000000 0000000000000000000000000000000000000000000000000100

Tabela 8: Zapis bitowy kolejnych liczb na przedziale $[2, 4]$

3.4 Wnioski

Liczby zmiennopozycyjne w arytmetyce Float64 są równomiernie rozmieszczone na badanych przedziałach z odpowiednimi krokami δ, które zależą od przedziału. Wyniki te są zgodne z teorią dotyczącą reprezentacji liczb zmiennopozycyjnych.

4 Zadanie 4

4.1 Opis problemu

Zadanie polegało na eksperymentalnym znalezieniu najmniejszej liczby dodatniej x, takiej że $1 + x \neq 1$ w arytmetyce Float64.

4.2 Rozwiązanie

Rozwiązanie zostało zaimplementowane w pliku `zadanie4.jl`.

4.3 Wyniki i interpretacja

Najmniejsza liczba dodatnia x, dla której $1 + x \neq 1$, wynosi $x = 1.000000057228997$.

4.4 Wnioski

Wynik ten jest zgodny z oczekiwaniami, ponieważ arytmetyka zmiennopozycyjna jest obarczona błędami zaokrągleń. Znalezienie takiej liczby ilustruje ograniczenia precyzji reprezentacji liczb zmiennopozycyjnych.

5 Zadanie 5

5.1 Opis problemu

Zadanie polegało na obliczeniu iloczynu skalarnego dwóch wektorów w typach Float32 oraz Float64 na cztery różne sposoby: w przód, w tył, dodając liczby dodatnie od największej i ujemne od najmniejszej oraz odwrotnie.

5.2 Rozwiązanie

Rozwiązanie zostało zaimplementowane w pliku `zadanie5.jl`.

5.3 Wyniki i interpretacja

Metoda	Wynik (Float32)	Wynik (Float64)
W przód	-0.4999443	$1.0251881368296672 \times 10^{-10}$
W tył	-0.4543457	$-1.5643308870494366 \times 10^{-10}$
Od największego do najmniejszego	-0.5	0.0
Od najmniejszego do największego	-0.5	0.0

Tabela 9: Wyniki iloczynu skalarnego dla różnych metod i typów danych

5.4 Wnioski

Kolejność dodawania ma istotny wpływ na wynik końcowy, zwłaszcza dla Float32. Metody dodawania od największego do najmniejszego oraz od najmniejszego do największego dały identyczne wyniki, co sugeruje większą stabilność numeryczną. Float64 wykazał większą precyzję i mniejsze błędy zaokrągleń.

6 Zadanie 6

6.1 Opis problemu

Zadanie polegało na obliczeniu wartości funkcji

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

oraz jej alternatywnej formy

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

dla bardzo małych wartości x i porównaniu wyników.

6.2 Rozwiązanie

Rozwiązanie zostało zaimplementowane w pliku `zadanie6.jl`.

6.3 Wyniki i interpretacja

x	$f(x)$	$g(x)$
8^{-1}	0.0077822185373186414	0.007782218537318706
8^{-2}	$1.2206286282867573 \times 10^{-4}$	$1.2206286282875901 \times 10^{-4}$
8^{-3}	$1.9073468138230965 \times 10^{-6}$	$1.907346813826566 \times 10^{-6}$
8^{-4}	$2.9802321943606103 \times 10^{-8}$	$2.9802321943606116 \times 10^{-8}$
8^{-5}	$4.656612873077393 \times 10^{-10}$	$4.6566128719931904 \times 10^{-10}$
8^{-6}	$7.275957614183426 \times 10^{-12}$	$7.275957614156956 \times 10^{-12}$
8^{-7}	$1.1368683772161603 \times 10^{-13}$	$1.1368683772160957 \times 10^{-13}$
8^{-8}	$1.7763568394002505 \times 10^{-15}$	$1.7763568394002489 \times 10^{-15}$
8^{-9}	0	$2.7755575615628914 \times 10^{-17}$

Tabela 10: Porównanie wyników funkcji $f(x)$ i $g(x)$ dla bardzo małych wartości x (fragment)

6.4 Wnioski

Alternatywna forma $g(x)$ jest bardziej stabilna numerycznie dla bardzo małych x. Funkcja $f(x)$ traci precyzję ze względu na odejmowanie liczb bardzo podobnych.

7 Zadanie 7

7.1 Opis problemu

Zadanie polegało na skorzystaniu ze wzoru na pochodną:

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

do obliczenia pochodnej funkcji

$$f(x) = \sin(x) + \cos(3x)$$

w punkcie $x_0 = 1$ dla różnych wartości h i porównaniu wyniku z wartością rzeczywistą pochodnej.

7.2 Rozwiązanie

Rozwiązanie zostało zaimplementowane w pliku `zadanie7.jl`. Stworzona została funkcja, będącą rzeczywistą pochodną funkcji f oraz funkcja obliczająca przybliżoną wartość pochodnej za pomocą podanego wzoru dla różnych wartości h.

7.3 Wyniki i interpretacja

h	Obliczona pochodna	Różnica	$1+h$
2^0	2.0179892252685967	1.9010469435800585	2.0
2^{-1}	1.8704413979316472	1.753499116243109	1.5
2^{-2}	1.1077870952342974	0.9908448135457593	1.25
2^{-3}	0.6232412792975817	0.5062989976090435	1.125
2^{-4}	0.3704000662035192	0.253457784514981	1.0625
2^{-5}	0.24344307439754687	0.1265007927090087	1.03125
2^{-28}	0.11694228649139404	$4.802855890773117 \times 10^{-9}$	1.0000000037252903
2^{-50}	0.0	0.11694228168853815	1.0000000000000009
2^{-51}	0.0	0.11694228168853815	1.0000000000000004
2^{-52}	-0.5	0.6169422816885382	1.0000000000000002
2^{-53}	0.0	0.11694228168853815	1.0

Tabela 11: Porównanie obliczonej pochodnej z wartością rzeczywistą dla różnych h

7.4 Wnioski

Wyniki pokazują, że w przeciwieństwie do wartości rzeczywistych, gdzie dokładność rośnie wraz ze zmniejszaniem h, w arytmetyce Float64 najmniejsza różnica wystąpiła dla $h = 2^{-28}$.