Sprawozdanie z Laboratorium 1
Obliczenia Naukowe

Marcel Musiałek

25 października 2025

1 Zadanie 1: Rozpoznanie arytmetyki

1.1 Część 1: Wyznaczanie epsilona maszynowego (*macheps*)

1.1.1 Opis problemu

Celem było wyznaczenie epsilona maszynowego (*macheps*), czyli najmniejszej liczby $x > 0$ takiej, że $1.0 + x \neq 1.0$. Zadanie wymagało porównania wartości uzyskanej iteracyjnie z wbudowaną funkcją Julii `eps(T)` oraz standardowymi stałymi z pliku `float.h` języka C.

1.1.2 Opis rozwiązania

Rozwiązanie polegało na zaimplementowaniu pętli, która iteracyjnie dzieliła wartość *macheps* (zainicjowaną jako 1.0 danego typu) przez 2.0. Pętla kontynuowała działanie tak długo, jak długo warunek $1.0 + (macheps/2.0) > 1.0$ był spełniony w arytmetyce danego typu. Ostatnia wartość *macheps*, dla której warunek ten nie był już prawdziwy, jest szukaną wartością.

1.1.3 Wyniki i interpretacja

Poniżej przedstawiono wyniki uzyskane z implementacji iteracyjnej oraz z funkcji wbudowanej.

```
--- Wyznaczanie Epsilona Maszynowego (macheps) ---
--- Typ: Float16 ---
Iteracyjnie:          9.76562500e-04
Wbudowane (eps):      9.76562500e-04
--- Typ: Float32 ---
Iteracyjnie:          1.19209290e-07
Wbudowane (eps):      1.19209290e-07
--- Typ: Float64 ---
Iteracyjnie:          2.22044605e-16
Wbudowane (eps):      2.22044605e-16
```

Interpretacja i porównanie z `float.h` (C): Wyniki obliczeń iteracyjnych są identyczne z funkcjami wbudowanymi Julii. Poniżej jawne zestawienie ze standardowymi stałymi C:

- **Float32 (Single):**
  - Nasz wynik: `1.19209290e-07`
  - Stała C `FLT_EPSILON`: `1.19209290e-07` (2^{-23})
  - *Zgodność: Tak*
- **Float64 (Double):**
  - Nasz wynik: `2.22044605e-16`
  - Stała C `DBL_EPSILON`: `2.22044605e-16` (2^{-52})
  - *Zgodność: Tak*

1.1.4 Wnioski

Obliczenia iteracyjne dały wyniki w pełni zgodne z wbudowanymi funkcjami Julii oraz standardowymi wartościami `FLT_EPSILON` i `DBL_EPSILON` ze **standardu IEEE 754**. Wartość ta definiuje *jednostkowy błąd zaokrąglenia* ($u = macheps/2$), który jest kluczowym parametrem w analizie błędów **obliczeń maszynowych** i pokazuje granicę precyzji względnej operacji.

1.2 Część 2: Wyznaczanie najmniejszej liczby dodatniej (η / MIN_{sub})

1.2.1 Opis problemu

Zadanie polegało na wyznaczeniu najmniejszej dodatniej liczby maszynowej η (tzw. MIN_{sub}). Należało użyć iteracyjnego dzielenia przez 2.0, aż do osiągnięcia 0.0, a następnie porównać wynik z `nextfloat(T(0.0))` oraz sprawdzić, czy `float.h` definiuje tę wartość.

1.2.2 Opis rozwiązania

Zastosowano pętlę iteracyjną, która dzieliła wartość η (zainicjowaną jako 1.0) przez 2.0. Pętla była kontynuowana, dopóki wynik dzielenia (`eta / 2.0`) był większy od 0.0. Ostatnia wartość η przed osiągnięciem zera (gdy `eta / 2.0` staje się równe 0.0) jest szukaną wartością.

1.2.3 Wyniki i interpretacja

Otrzymane wyniki z algorytmu iteracyjnego oraz funkcji wbudowanej:

```
--- Wyznaczanie Najmniejszej Liczby Dodatniej (eta / MIN_sub) ---
--- Typ: Float16 ---
Iteracyjnie:                  5.96046448e-08
Wbudowane (nextfloat(0.0)):   5.96046448e-08
--- Typ: Float32 ---
Iteracyjnie:                  1.40129846e-45
Wbudowane (nextfloat(0.0)):   1.40129846e-45
--- Typ: Float64 ---
Iteracyjnie:                  4.94065646e-324
Wbudowane (nextfloat(0.0)):   4.94065646e-324
```

Interpretacja i porównanie z `float.h` (C): Wyniki iteracyjne są identyczne z wartościami `nextfloat(T(0.0))`.

- **Float32 (Single):**
  - Nasz wynik (η): `1.40129846e-45`
  - Stała C `float.h`: *Brak* (standard C definiuje tylko `FLT_MIN`, patrz Część 3)
- **Float64 (Double):**
  - Nasz wynik (η): `4.94065646e-324`
  - Stała C `float.h`: *Brak* (standard C definiuje tylko `DBL_MIN`, patrz Część 3)

1.2.4 Wnioski

Eksperyment potwierdził istnienie **liczb subnormalnych (zdenormalizowanych)**, kluczowego elementu **standardu IEEE 754**. Wyznaczona η (MIN_{sub}) reprezentuje mechanizm "łagodnego niedomiaru"(gradual underflow). W **obliczeniach maszynowych** pozwala to uniknąć gwałtownego "przeskoku"do zera, co jest krytyczne dla stabilności algorytmów operujących na bardzo małych wartościach.

1.3 Część 3: Analiza `floatmin` vs MIN_{nor} (i `float.h`)

1.3.1 Opis problemu

Celem było sprawdzenie, co zwraca funkcja `floatmin(T)` i jaki ma związek z MIN_{nor} (najmniejszą liczbą znormalizowaną) oraz ze stałymi `FLT_MIN` i `DBL_MIN` z `float.h`.

1.3.2 Opis rozwiązania

W tej części nie implementowano nowego algorytmu iteracyjnego. Wywołano jedynie wbudowaną funkcję Julii `floatmin(T)` oraz funkcję `find_eta(T)` (zdefiniowaną w poprzedniej części) w celu ich bezpośredniego porównania w wygenerowanym wydruku.

1.3.3 Wyniki i interpretacja

Wyniki porównania funkcji `floatmin(T)` z obliczoną wcześniej η (MIN_{sub}):

```
--- Badanie floatmin(T) ---
floatmin(Float16):     6.10351562e-05 (vs eta: 5.96046448e-08)
floatmin(Float32):     1.17549435e-38 (vs eta: 1.40129846e-45)
floatmin(Float64):     2.22507386e-308 (vs eta: 4.94065646e-324)
```

Interpretacja i porównanie z `float.h` (C): Wartości `floatmin(T)` są znacznie większe niż η (MIN_{sub}). Porównanie ze stałymi C:

- **Float32 (Single):**
  - Nasz wynik (`floatmin`): `1.17549435e-38`
  - Stała C `FLT_MIN`: `1.17549435e-38`
  - *Zgodność: Tak*
- **Float64 (Double):**
  - Nasz wynik (`floatmin`): `2.22507386e-308`
  - Stała C `DBL_MIN`: `2.22507386e-308`
  - *Zgodność: Tak*

1.3.4 Wnioski

Eksperyment jasno pokazuje fundamentalne rozróżnienie w **standardzie IEEE 754**:

1. η (MIN_{sub}): Najmniejsza liczba subnormalna (obszar *gradual underflow*).
2. `floatmin(T)` (MIN_{nor}): Najmniejsza liczba **znormalizowana**.

Jak widać z jawnego porównania, to właśnie MIN_{nor} (a nie MIN_{sub}) jest zdefiniowane w standardzie C (`FLT_MIN`, `DBL_MIN`). W **obliczeniach maszynowych** MIN_{nor} wyznacza początek zakresu, w którym liczby mają pełną precyzję względną.

1.4 Część 4: Wyznaczanie największej liczby skończonej (MAX)

1.4.1 Opis problemu

Wyznaczenie największej skończonej liczby maszynowej (MAX). Należało znaleźć największą potęgę dwójki $P = 2^{E_{max}}$ (mnożąc do `Inf`), a następnie obliczyć $MAX = P \times (2.0 - macheps)$. Wynik należało porównać z `floatmax(T)` oraz stałymi z `float.h`.

1.4.2 Opis rozwiązania

Algorytm składał się z dwóch kroków. Najpierw, w pętli mnożono 1.0 przez 2.0 aż do osiągnięcia nieskończoności (`Inf`), aby znaleźć największą potęgę dwójki $P = 2^{E_{max}}$. Następnie obliczono MAX ze wzoru $MAX = P \times (2.0 - macheps)$, wykorzystując *macheps* obliczony w Części 1.

1.4.3 Wyniki i interpretacja

Wyniki iteracyjnego obliczania MAX w porównaniu z funkcją wbudowaną:

```
--- Wyznaczanie Największej Liczby Skończonej (MAX) ---
--- Typ: Float16 ---
Iteracyjnie:          6.55040000e+04
Wbudowane (floatmax):   6.55040000e+04
--- Typ: Float32 ---
Iteracyjnie:          3.40282347e+38
Wbudowane (floatmax):   3.40282347e+38
--- Typ: Float64 ---
Iteracyjnie:          1.79769313e+308
Wbudowane (floatmax):   1.79769313e+308
```

Interpretacja i porównanie z `float.h` (C): Wyniki iteracyjne są identyczne z wartościami `floatmax(T)`. Porównanie ze stałymi C:

- **Float32 (Single):**
  - Nasz wynik: `3.40282347e+38`
  - Stała C `FLT_MAX`: `3.40282347e+38`
  - *Zgodność: Tak*
- **Float64 (Double):**
  - Nasz wynik: `1.79769313e+308`
  - Stała C `DBL_MAX`: `1.79769313e+308`
  - *Zgodność: Tak*

1.4.4 Wnioski

Wyznaczona wartość MAX reprezentuje górną granicę zakresu liczb znormalizowanych. Jawne porównanie potwierdza pełną zgodność ze **standardem IEEE 754** oraz stałymi C (`FLT_MAX`, `DBL_MAX`). W **obliczeniach maszynowych**, przekroczenie tej wartości skutkuje "nadmiarem"(overflow) i jest sygnalizowane wartością `Inf`. Zrozumienie tego limitu jest niezbędne przy projektowaniu stabilnych numerycznie algorytmów.

2 Zadanie 2: Obliczanie macheps metodą Kahana

2.1 Weryfikacja formuły Kahana

2.1.1 Opis problemu

Profesor William Kahan zasugerował, że epsilon maszynowy (*macheps*) można wyznaczyć w arytmetyce zmiennoprzecinkowej poprzez obliczenie wyrażenia: $3 \times (4/3 - 1) - 1$.

Celem tego zadania była eksperymentalna weryfikacja tego stwierdzenia w języku Julia dla typów `Float16`, `Float32` i `Float64`. Porównaliśmy wynik tej formuły z wartością zwracaną przez wbudowaną funkcję `eps(T)`.

2.1.2 Opis rozwiązania

Zaimplementowano funkcję, która krok po kroku wykonuje obliczenia formuły Kahana. Kluczowe było, aby wszystkie stałe (1.0, 3.0, 4.0) oraz wszystkie operacje pośrednie były wykonywane ściśle w arytmetyce badanego typu (`Float16`, `Float32`, `Float64`). Krok po kroku obliczono $fl(4/3)$, następnie $fl(fl(4/3) - 1)$, potem $fl(3 \times \dots)$, i na końcu odjęto 1.0, aby wyizolować błąd. Pozwoliło to na precyzyjne uchwycenie błędów zaokrągleń specyficznych dla każdej precyzji.

2.1.3 Wyniki i interpretacja

Poniżej przedstawiono wyniki uzyskane z implementacji formuły Kahana oraz wartości z funkcji wbudowanej.

```
--- Sprawdzanie formuły Kahana dla macheps ---
--- Typ: Float16 ---
Wynik z formuły Kahana:     -9.7656250000e-04
Wbudowane (eps(T)):          9.7656250000e-04
Czy równe eps(T)?            false
Czy równe -eps(T)?           true
--- Typ: Float32 ---
Wynik z formuły Kahana:     1.1920928955e-07
Wbudowane (eps(T)):          1.1920928955e-07
Czy równe eps(T)?            true
Czy równe -eps(T)?           false
--- Typ: Float64 ---
Wynik z formuły Kahana:     -2.2204460493e-16
Wbudowane (eps(T)):          2.2204460493e-16
Czy równe eps(T)?            false
Czy równe -eps(T)?           true
```

Interpretacja: Eksperyment dał zaskakujący i bardzo pouczający wynik. Wartość bezwzględna formuły Kahana jest zawsze równa `eps(T)`, jednak znak wyniku zależy od typu zmiennoprzecinkowego.

- **Float16:**
  - Wynik Kahana: `-9.7656250000e-04`
  - Wbudowane `eps(T)`: `9.7656250000e-04`
  - Zgodność: `kahan_val == -eps(T)` (Prawda)
- **Float32:**
  - Wynik Kahana: `1.1920928955e-07`
  - Wbudowane `eps(T)`: `1.1920928955e-07`
  - Zgodność: `kahan_val == eps(T)` (Prawda)
- **Float64:**
  - Wynik Kahana: `-2.2204460493e-16`
  - Wbudowane `eps(T)`: `2.2204460493e-16`
  - Zgodność: `kahan_val == -eps(T)` (Prawda)

2.1.4 Wnioski

Stwierdzenie Kahana jest słuszne co do *wartości bezwzględnej* – formuła ta doskonale izoluje błąd zaokrąglenia o wielkości *macheps*. Znak wyniku jest fascynującą ilustracją subtelności reguł zaokrąglania w **standardzie IEEE 754**.

Kluczowa jest pierwsza operacja: $fl(4/3)$. Liczba $4/3$ ma nieskończone rozwinięcie binarne: $1.01010101\ldots_2$. Standard **IEEE 754** stosuje zaokrąglanie do najbliższej wartości ("round-to-nearest-ties-to-even").

1. **Przypadek `Float16` ($p = 11$ bitów) i `Float64` ($p = 53$ bity):** Dla obu tych precyzji, odcięta część rozwinięcia binarnego $1.0101\ldots_2$ jest mniejsza niż połowa jednostki na ostatnim miejscu (ULP). Liczba $4/3$ jest zaokrąglana **w dół**. Analiza błędu pokazuje, że $fl(4/3) = 4/3 - \epsilon_1$ (błąd początkowy jest ujemny), co po kolejnych operacjach i końcowej anulacji daje wynik $-macheps$.
2. **Przypadek `Float32` ($p = 24$ bity):** Dla tej konkretnej precyzji, odcięta część rozwinięcia jest większa niż połowa ULP. Liczba $4/3$ jest zaokrąglana **w górę**. Analiza błędu pokazuje, że $fl(4/3) = 4/3 + \epsilon_1$ (błąd początkowy jest dodatni), co po propagacji błędu i końcowej anulacji daje wynik $+macheps$.

Eksperyment ten pokazuje, że kierunek zaokrąglenia w **obliczeniach maszynowych** zależy od konkretnej liczby bitów precyzji, co może prowadzić do pozornie sprzecznych (ale poprawnych) wyników dla różnych typów zmiennoprzecinkowych.

3 Zadanie 3: Błędy anulacji i błąd względny

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

3.1.1 Opis problemu

Zadanie polegało na obliczeniu wartości funkcji $f(x) = \sqrt{x^2+1} - 1$ dla x malejących do zera ($x = 8^{-1}, 8^{-2}, \ldots, 8^{-k}$). Celem była obserwacja utraty precyzji (błędu anulacji) oraz porównanie wyników z matematycznie równoważną, ale stabilniejszą numerycznie postacią $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$.

3.1.2 Opis rozwiązania

Obie funkcje, $f(x)$ i $g(x)$, zostały zaimplementowane i wywołane dla typów `Float32` i `Float64`. Jako "dokładną"wartość referencyjną y przyjęto wynik funkcji $g(x)$ obliczony w precyzji `Float64`. Następnie obliczono błąd względny $\frac{|fl(f(x))-y|}{|y|}$ oraz $\frac{|fl(g(x))-y|}{|y|}$ dla obu typów.

3.1.3 Wyniki i interpretacja

Poniżej kluczowe obserwacje z wygenerowanych wyników (dla czytelności pominięto pełne tabele).
Dla `Float32`:

- **Funkcja** $f(x)$ **(niestabilna):** Już przy $x \approx 1.5 \times 10^{-4}$ błąd względny przekracza 10%. Dla $x \approx 9.5 \times 10^{-5}$ wynik $f(x)$ staje się równy **zero**, a błąd względny wynosi 100%. Jest to klasyczny przykład **błędu anulacji** – gdy $\sqrt{x^2+1} \approx 1$, odejmowanie 1 prowadzi do katastrofalnej utraty cyfr znaczących.

- **Funkcja** $g(x)$ **(stabilna):** Błąd względny dla $g(x)$ utrzymuje się na poziomie błędu maszynowego ($\approx 10^{-8}$) przez cały zakres testów, nawet gdy $f(x)$ zwraca już zero.

Dla `Float64`:

- **Funkcja** $f(x)$ **(niestabilna):** Zjawisko anulacji również występuje, ale znacznie później, co jest oczekiwane przy większej precyzji. Błąd względny zaczyna gwałtownie rosnąć dla $x < 10^{-8}$. Dla $x \approx 2.2 \times 10^{-8}$ błąd sięga ok. 15%, a dla $x \approx 1.7 \times 10^{-9}$ $f(x)$ zwraca **zero** (błąd 100%).

- **Funkcja** $g(x)$ **(stabilna):** Błąd względny pozostaje na poziomie *macheps* dla `Float64` ($\approx 10^{-16}$) aż do granicy liczb subnormalnych.

3.1.4 Wnioski

Eksperyment dobitnie pokazał, jak krytyczny w **obliczeniach maszynowych** jest dobór algorytmu. Matematyczna równoważność $f(x) = g(x)$ nie implikuje równoważności numerycznej. Postać $f(x)$ cierpi na katastrofalny błąd anulacji (odejmowanie bardzo bliskich sobie liczb), podczas gdy postać $g(x)$, uzyskana przez przekształcenie (wzorami skróconego mnożenia), jest numerycznie stabilna, ponieważ eliminuje problematyczne odejmowanie.