Obliczenia naukowe lista 1

Stanisław Tomkowiak

27 listopada 2024

Zadanie 1

Macheps

Epsilon maszynowy czyli właśnie macheps jest to najmniejsza liczba taka, żę $1 + macheps > 1$. Dzięki temu epsilonowi określamy precyzję arytmetyki. Błąd zaokrąglenia liczby wynosi dokładnie $\frac{1}{2}$ macheps.

Wyznaczenie wartości epsilonu maszynowego metodą iteracyjną polega na dodawaniu do liczby 1.0 w wybranej precyzji coraz mniejszych wartości, dopóki liczba jest większa od 1.0. Jako pierwszą liczbę ustalamy macheps=1.0 i w każdej iteracji dzielimy go przez 2. Pętla zatrzyma się na wartości macheps.

Obliczanie maszynowego epsilonu dla typu liczbowego T:

```
function MACHEPS(T)
    macheps ← T(1.0)                          ▷ Inicjalizujemy macheps jako 1.0 dla typu T
    while T(1.0) + macheps/T(2.0) > T(1.0) do
        macheps ← macheps/T(2.0)              ▷ Dzielimy macheps przez 2, dopóki sumowanie go z T(1.0) jest
    większe od 1.0
    end while
    return macheps                            ▷ Zwracamy obliczoną wartość macheps
end function
```

Poniżej przedstawiam wyniki z dokładnością do trzech cyfr znaczących dla trzech różnych metod wyznaczenia epsilonu:

	Float16	Float32	Float64
wyniki iteracyjne	0.000977	$1.19 \cdot 10^{-7}$	$2.22 \cdot 10^{-16}$
wyniki uzyskane za pomocą funkcji eps()	0.000977	$1.19 \cdot 10^{-7}$	$2.22 \cdot 10^{-16}$
wartości w float.h	brak	$1.19 \cdot 10^{-7}$	$2.22 \cdot 10^{-16}$

Wyniki dla tych trzech metod są takie same. Znaczy to, że iteracyjny sposób wyznaczenia epsilonu maszynowego jest skuteczny.

Eta

Eta jest to najmniejsza liczba większa od zera maszynowego. W zadaniu należało obliczyć iteracyjnie wartość liczby eta dla wszystkich typów zmiennopozycyjnych zgodnych ze standardem IEEE 754. Dodatkowo musieliśmy zbadać związek tej liczby z liczbą MIN_{sub}.

Wyznaczenie tych wartości jest analogiczne do wyznaczania macheps. Dzielimy liczbę 1.0 przez 2 do czasu aż jest ona większa od 0.0. Poniżej przedstawiam pseudokod:

Obliczanie liczby eta dla typu liczbowego T

```
function ETA(T)
    eta ← T(1.0)                              ▷ Inicjalizujemy eta jako 1.0 dla typu T
    while eta/T(2.0) > T(0.0) do
        eta ← eta/T(2.0)                      ▷ Dzielimy eta przez 2, dopóki jest większa od 0.0
    end while
    return eta                                ▷ Zwracamy obliczoną wartość macheps
end function
```

Poniżej przedstawiam wyniki dwóch różnych metod wyznaczenia eta:

	Float16	Float32	Float64
wyniki iteracyjne	6.0e-8	1.0e-45	5.0e-324
wyniki uzyskane za pomocą funkcji nextfloat(0.0)	6.0e-8	1.0e-45	5.0e-324

Wyniki dla tych dwóch metod są takie same. Znaczy to, że iteracyjny sposób wyznaczenia liczby eta jest skuteczny.

Związek liczby eta z MIN_{sub}. MIN_{sub} jest najmniejszą liczbą większą od 0.0 w formie nieznormalizowanej. Po obliczeniu liczby MIN_{sub} dla badanych typów jest ona równa liczbie eta. Wyniki funkcji minfloat(Float32) i minfloat(Float64) prezentują się następująco:

- floatmin32: 1.1754944e-38
- floatmin64: 2.2250738585072014e-308

Są one zatem wiele większe od iteracyjnej próby, wartości te są najmniejszymi wartościami większymi od 0.0 znormalizowanymi.

Max

Największa wartość jaką można otrzymać w standardzie IEEE 754. Otrzymuję ją za pomocą algorytmu w którym mnożymy liczbę previous float(1.0) razy 2 tak długo aż liczba jest różna od wartości infinity. Poniżej przedstawiam pseudokod:

Obliczanie maksymalnej wartości reprezentowalnej dla typu liczbowego T

```
function MAX(T)
    max ← prevfloat(T(1.0))        ▷ Inicjalizujemy max jako największą liczbę mniejszą niż 1.0 dla typu T
    while not isinf(max × T(2.0)) do
        max ← max × T(2.0)         ▷ Mnożymy max przez 2, dopóki nie osiągnie nieskończoności
    end while
    return max                     ▷ Zwracamy obliczoną maksymalną wartość
end function
```

Poniżej przedstawiam wyniki badań:

	Float16	Float32	Float64
wyniki iteracyjne	$6.55 \cdot 10^4$	$3.40 \cdot 10^{38}$	$1.80 \cdot 10^{308}$
wynikiuzyskane za pomocą funkcji floatmax()	$6.55 \cdot 10^4$	$3.40 \cdot 10^{38}$	$1.80 \cdot 10^{308}$
wartości z raportu	brak danych	$3.40 \cdot 10^{38}$	$1.80 \cdot 10^{308}$

Wyniki są takie same. Znaczy to, że iteracyjna forma wyznaczenia maksymalnej wartości jest prawidłowa(zgadza się z danymi z raportu oraz wynikami funkcji floatmax).

Zadanie 2

Wzór wymyślony przez Kahana: $3(\frac{4}{3}-1)-1$ po obliczeniu zwraca wyniki zgodne z machepsem danego typu co do znaku. Można uznać je za poprawne ponieważ w standardzie IEEE 754 pomnożenie przez -1 to jedynie negacja pierwszego bitu. Jednak aby uznać wzór kahana za w pełni poprawny należałoby zapisać go w formie:

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

Zadanie 3

Sprawdzenie rozmieszczenia liczb w standardzie IEEE 754 arytmetyce Float64. Z powodów bardzo dużej liczby liczb sprawdzamy tylko mały wycinek tych liczb z przedziału. Dla zbadanych wartości wynika, że przedziale [1,2] liczby zmiennopozycyjne są rozmieszczone równomiernie. Krok tego przedziału jest równy $\delta = 2^{-52}$. Zatem każda liczba występująca w tym przedziale może być przedstawiona za pomocą wzoru $x = 1 + k\delta$ dla $k = 1, 2, ..., 2^{-52} - 1$. Badając jednak inne przedstawione w zadaniu

przedziału obserwujemy iż kroki są inne. Przedział [0.5,1] ma krok równy $\delta = 2^{-53}$. Przedział [2,4] ma krok równy $\delta = 2^{-51}$.

W badanym standardzie w reprezentacji bitowej obserwujemy,że przedziały $[2^k, 2^{k+1}]$ różnią się tylko mantysą. W każdym takim przedziale znajduje się 2^t równo rozmieszczonych liczb. Widzimy zatem, że im mniejsza cecha w danym przedziale tym bliższe sobie liczby jesteśmy w stanie zaobserwować.

Zadanie 4

Najmniejsza liczba w arytmetyce Float64 w przedziale (1,2) taka, że $x * \frac{1}{x} \neq 1$ to 1.000000057228997. Wyznaczyć ją można stosując metody poznane już wcześniej. Mianowicie zaczynając od następnej liczby po 1.0 wykonywać to działanie zwiększając liczbę o wartość macheps do czasu aż wartość iloczynu będzie różna od 1.0. Warto także sprawdzić jaki wynik otrzymamy po wykonaniu tego działania z tą liczbą. Jest ono równe: 0.9999999999999999. Wnioski z przeprowadzenia tego badania są takie, że ograniczenia w arytmetyce zmiennopozycyjnej mogą doprowadzać do błędów w wynikach przez co mogą zostać złamane pewne własności matematyczne.

Zadanie 5

Zadanie polegało na obliczeniu iloczynu skalarnego dwóch podanych wektorów:

$$\begin{aligned} x &= [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957] \\ y &= [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]. \end{aligned}$$

Uzyskać wynik mieliśmy na cztery różne sposoby dla arytmetyk Float32 i Float64. Poniżej przedstawiam wyniki tych eksperymentów:

	Float32	Float64
prawidłowa wartość	-1.00657107000000·10^{-11}	-1.00657107000000·10^{-11}
"w przód"	-0.4999443	$1.0251881368296672 \cdot 10^{-10}$
"w tył"	-0.4543457	$-1.5643308870494366 \cdot 10^{-10}$
"malejąco"	-0.5	0.0
"rosnąco"	-0.5	0.0

Żaden ze sposobów nie jest skuteczny. Szczególnie duże błędy pojawiają się w precyzji single. Wyniki przedstawione w tabeli nie pojawiły się bez powodu; tak duże błędy wynikają z tego, że wektory te są prawie prostopadłe do siebie. Powstają przez to duże błędy w obliczeniach. Dodatkowo, dzięki temu zadaniu widać, iż kolejność wykonywania działań w komputerze ma znaczenie. Na błędy wpływa nie tylko arytmetyka, ale także właśnie kolejność działań.

Zadanie 6

Zadanie polegało na obliczeniu wartości funkcji:

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

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

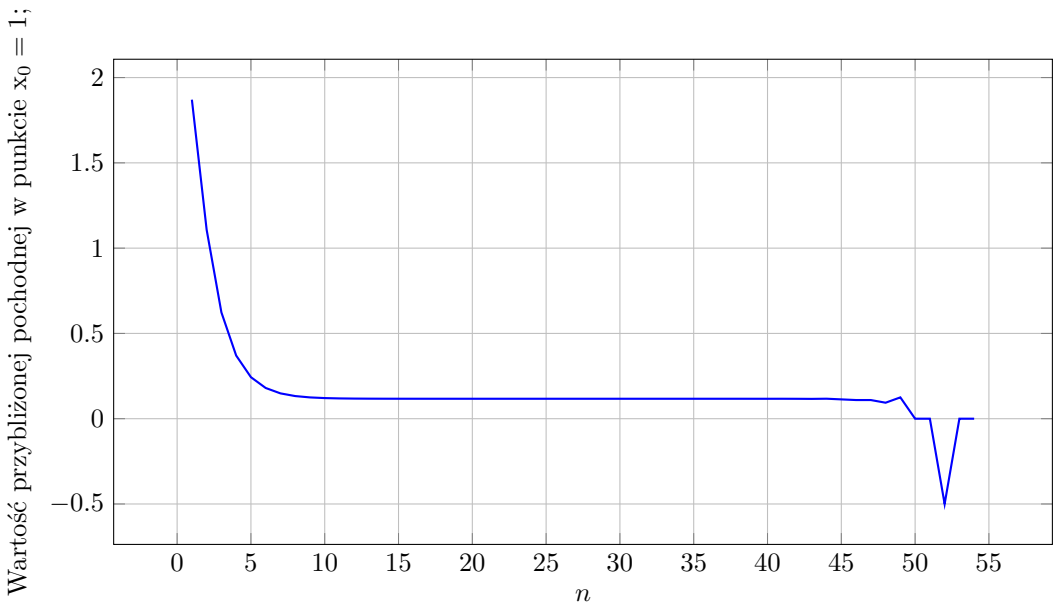
Dla różnych wartości argumentu $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$. Warto również zaznaczyć, że z matematycznego sensu te dwie funkcje są sobie równe. Należało policzyć to w precyzji Float64. Funkcja $f(x)$ jest równa 0.0 dla $x = 8^{-9}$, natomiast funkcja $g(x)$ jest równa 0.0 dla $x = 8^{-179}$. Wynika z tego, że wyniki funkcji $g(x)$ są dużo bardziej wiarygodne. Funkcja ta nigdy dla argumentów większych od 0 nie powinna osiągać wartości 0.0. Taka rozbieżność wyników wynika z tego, że korzystając z funkcji $f(x)$ odejmujemy od siebie bardzo bliskie liczby przez co otrzymujemy duży błąd podczas obliczeń. W sytuacji użycia funkcji $g(x)$ usuwamy ten problem. Przekształcając funkcje aby uniknąć odejmowania bliskich sobie liczb pozwala zwiększyć wiarygodność wyników, oraz zmniejszyć błędy podczas obliczeń.

Zadanie 7

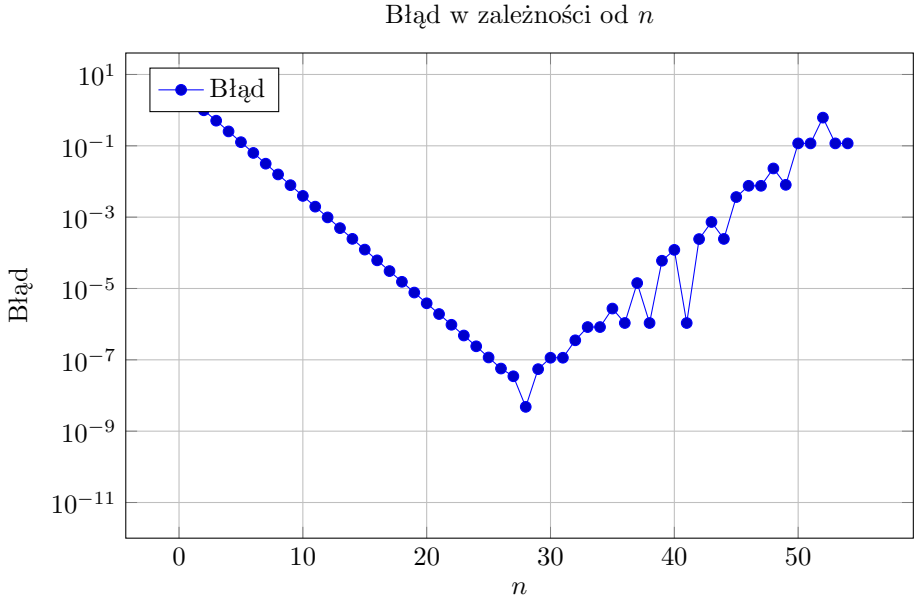
W zadaniu mamy policzyć przybliżoną wartość pochodnej w punkcie $x_0 = 1$. Zostaje nam do tego zadania dostarczony wzór:

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

gdzie $h = 2^{-n}$ $(n = 0, 1, 2, ..., 54)$. Poniżej przedstawiam wyniki na wykresie:



Dodatkowo przedstawiam wyniki pomiaru błędu względem prawidłowej wartości pochodnej tej funkcji:



Rysunek 1: Wykres błędu w zależności od liczby iteracji n.

Możemy zaobserwować, że najbliższe przybliżenie mamy dla wartości $n = 28$. Dalsze zmniejszanie h powoduje wzrost błędu. Wynika to z arytmetyki Float64 oraz odejmowania pojawiającego się w tym wzorze. Dodatkowo, po bliższym przyjrzeniu się wartościom $h + 1$ widzimy, że dla $n = 53$ $h + 1 = 1$. Przez ograniczenia arytmetyki w pewnym momencie zaczynamy podczas odejmowania ucinać coraz

to większe fragmenty h przez co tracimy na dokładności przybliżenia. Wniosek wynikający z tego zadania jest taki, że mimo, iż zmniejszamy coraz bardzie h to w pewnym momencie zaczynamy się oddalać od prawdziwej wartości. Wydawałoby się ze zmniejszanie tak h może nas przybliżać jednak przez ograniecznia arytmetyki ucinamy część mantysy h co oddala nas od wyników prawidłowych.