# Obliczenia naukowe
# Lista1

Stanisław Woźniak

# 1 Zadanie 1

## 1.1 Macheps - Epsilon Maszynowy

### 1.1.1 Problem

Należało obliczyć iteracyjnie epsilon maszynowy (*macheps*) w arytmetykach Float16, Float32 oraz Float64. Macheps jest to najmniejsza liczba $\epsilon > 0$ , taka że $1.0 + \epsilon > 1.0$ .

### 1.1.2 Wyniki

| | Float16 | Float32 | Float64 |
|---|---|---|---|
| wyliczone | 0.000977 | $1.1920929 * 10^{-7}$ | $2.220446049250313 * 10^{-16}$ |
| systemowe | 0.000977 | $1.1920929 * 10^{-7}$ | $2.220446049250313 * 10^{-16}$ |
| float.h | - | $1.1920929 * 10^{-7}$ | $2.220446049250313 * 10^{-16}$ |

### 1.1.3 Wnioski

Po wyliczeniu *macheps* mozemy zauważyć, że uzyskane wyniki są takie same jak systemowe wartości oraz wartości z pliku nagłówkowego float.h.

Również możemy wywnioskować, że w zależności od precyzji arytmetyki zmienia się *macheps*. Zależnosc jest taka, że im lepsza precyzja tym mniejszy jest epsilon maszynowy.

## 1.2 Eta

### 1.2.1 Problem

Problem polegał na tym, żeby obliczyć wartość $\eta > 0$, która jest różnicą pomiędzy dwoma najbliższymi wartościami w danej arytmetyce. Obliczenia należało wykonywać w arytemtykach Float16, Float32, Float64.

### 1.2.2 Wyniki

| | Float16 | Float32 | Float64 |
|---|---|---|---|
| wyliczone | $6.0 * 10^{-8}$ | $1.0 * 10^{-45}$ | $5.0 * 10^{-324}$ |
| systemowe | $6.0 * 10^{-8}$ | $1.0 * 10^{-45}$ | $5.0 * 10^{-324}$ |
| $MIN_{nor}$ | - | $1.2 * 10^{-38}$ | $2.2 * 10^{-308}$ |
| $MIN_{sub}$ | - | $1.4 * 10^{-45}$ | $4.9 * 10^{-324}$ |
| floatmin() | $6.104 * 10^{-5}$ | $1.1754944 * 10^{-38}$ | $2.22507385850720 * 10^{-308}$ |

### 1.2.3 Wnioski

Pwyższe wyniki pokazują najmniejszą możliwą liczbę zapisaną w danej arytmetyce. Można to udowodnić za pomocą reprezentacji bitowej liczby gdzie w arytmetyce podwójnej dokładności liczba $\eta$ jest zapisana jako same zera oraz jedna jedynka na najmniej znaczacym bicie.

Jeśli byśmy porównywali wyliczone wartości z wartością $MIN_{sub}$ widzimy porównywalne wartości, co oznacza, że obliczone przez nas minima są zdenormalizowane.

Natomiast porównując $MIN_{nor}$ z zwróconą wartością floatmin(), także uzyskujemy wręcz identyczne wartości. Oznacza to, że funkcja floatmin() zwraca znormalizowane liczby.

## 1.3 Max Float - Największa wartość

### 1.3.1 Problem

Należało polczyć iteracyjne najwiekszą wartość (MAX) dla trzech arytmetyk (Float16, Float32, Float64) oraz porównanie ich z wartościami zwracanymi przez funkcję floatmax().

### 1.3.2 Wyniki

| | Float16 | Float32 | Float64 |
|---|---|---|---|
| wyliczone | $6.55 * 10^{4}$ | $3.4028235 * 10^{38}$ | $1.7976931348623157 * 10^{308}$ |
| systemowe | $6.55 * 10^{4}$ | $3.4028235 * 10^{38}$ | $1.7976931348623157 * 10^{308}$ |
| float.h | - | $3.40282347 * 10^{38}$ | $1.7976931348623157 * 10^{308}$ |

### 1.3.3 Wnioski

Dokładniej przyglądając się wynikom, możemy zauważyć, że wyniki wyliczone iteracyjnie pokrywają się z wynikami systemowymi oraz wynikami z pliku nagłówkowego float.h.

# 2 Zadanie 2 - Wzór na macheps

## 2.1 Problem

Należało stwierdzić eksperymentalnie prawdziwość wzoru Kahana na epsilon maszynowy (macheps), który jest według niego opisany wzorem:

$$3 * (\frac{4}{3} - 1) - 1$$

## 2.2 Wyniki

Porównanie wyników z poprawnym epsilonem maszynowym:

| | Wyliczony według wzoru | poprawny |
|---|---|---|
| Float16 | -0.000977 | 0.000977 |
| Float32 | $1.1920929 * 10^{-7}$ | $1.1920929 * 10^{-7}$ |
| Float64 | $-2.220446049250313 * 10^{-16}$ | $2.220446049250313 * 10^{-16}$ |

## 2.3 Wnioski

Jak można zobaczyć po wynikach, część z nich ma odwrotny znak od prawidłowej wartości. Dzieje się tak z powodu $\frac{4}{3}$ ponieważ w arytmetyce zmiennopozycyjnej Float64 nie ma dokładnej reprezentacji tejże wertości, z powodu czego jest błąd znaku w wyniku.

# 3 Zadanie 3 - Rozmieszczenie liczb zmiennopozycyjnych

## 3.1 Problem

Należało sprawdzić eksperymentalnie własność liczb w arytmetyce Float64 w przedziale [1, 2], że każda z nich może być przedstawiona wzorem:

$$x = 1 + k * \delta$$

gdzie $k = 1, 2, ...2^{52} - 1$ oraz $\delta = 2^{-52}$

## 3.2 Wynik

Wyniki wychodzą zgodne dla podanego wzoru porównywanego z funkcją nextfloat. Przy porównaniu liczb w bitach możemy również zauważyć, że wyniki są takie same w obu przypadkach.

### 3.3 Wnioski

Podany wzór jest poprawny, gdyż liczba $\delta$ jest reprezentowana jako jeden najmniej znaczący bit w arytmetyce Float64. Mnożąc po kolei przez każdą liczbę k uzyskujemy każdą możliwą reprezentację 52 najmniej znaczacych bitów, czyli mantysy.

Natomiast liczby w przedzale $[\frac{1}{2}, 1]$ są rozmieszczone z różnicą $\delta = 2^{-53}$, a w przedziale $[2, 4]$ sa rozmieszczone z różnicą $\delta = 2^{-51}$, ponieważ liczba ostateczna jest w reprezentacji $mantysa * 2^{cecha}$. Wynikiem czego są różnice pomiędzy $\delta$ w przedziale $[\frac{1}{2}, 1]$ a $\delta$ w przedziale $[1, 2]$ lub $[2, 4]$.

## 4 Zadanie 4

### 4.1 Problem

Należało znaleźć liczbę w arytmetyce Float64 z przedziału $1 < x < 2$, taką, że $x * \frac{1}{x} \neq 1$.

### 4.2 Wynik

| najmniejsza | największa |
|---|---|
| 1.000000057228997 | 1.9999999850988384 |

### 4.3 Wnioski

Z powyższych wyników widzimy, że istnieją liczby, które spełniają tą nierówność. Dzieje się to dlatego, gdyż działanie dzielenia nie daje za każdym razem dokładnej wartości. Ponieważ w ograniczonej ilości bitów nie jest możliwe zapisanie dokładnej wartości każdej liczby.

## 5 Zadanie 5 - Iloczyn skalarny dwóch wektorów

### 5.1 Problem

Przy pomocy 4 różnych algorytmów do policzenia iloczynu skalarnego, należało porównać wyniki do prawidłowej wartości: $-1.00657107000000 * 10^{-11}$.

Dane wektory:

$$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$

$$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$$

## 5.2 Algorytmy i ich wyniki

1. Algorytm liczący sumę kolejnych iloczynów wartości wektora
2. Algorytm liczący sumę kolejnych iloczynów wartości wektora zaczynając od ostatniego
3. Algorytm po wyliczeniu iloczynów dodaje wyniki dodatnie od największego do najmniejszego, następnie dodaje osobno wyniki ujemne od najmniejszego do największego. Ostatecznie obie składowe sumują się w końcowy wynik.
4. Algorytm po wyliczeniu iloczynów dodaje wyniki dodatnie od najmniejszego do największego, następnie dodaje osobno wyniki ujemne od największego do najmniejszego. Ostatecznie obie składowe sumują się w końcowy wynik.

Wyniki:

| algorytm | wynik dla Float32 | wynik dla Float64 |
|---|---|---|
| pierwszy | -0.4999443 | $1.0251881368296672 * 10^{-10}$ |
| drugi | -0.4543457 | $-1.5643308870494366 * 10^{-10}$ |
| trzeci | -0.5 | 0.0 |
| czwarty | -0.5 | 0.0 |

poprawna wartość:

$$-1.00657107000000 * 10^{-11}$$

## 5.3 Wnioski

Patrząc na prawidłowy wynik widzimy, że podane wektory są niemal prostopadłe. Natomiast patrząc na wyniki algorytmów możemy zauważyć, że różnią się one od prawidłowej wartości. Przy mniejszej dokładności (Float32) wyniki każdego algorytmy sa do siebie zbliżone, ale dalekie od prawidłowej wartości. Przy większej dokładności (Float64) wyniki są bliższe prawidłowej wartości, lecz jednak nadal odbiegają od niej. Dzieje się to dlatego, że przy dodawaniu dwóch wartości o dużej różnicy między sobą, większa wartość "pochłania"mniejszą wartość co daje niepoprawny wynik.

# 6 Zadanie 6 - Porównanie dwóch takich samych funkcji

## 6.1 Problem

Podane zostały dwie funkcje, które w praktyce są identyczne. W arytmetyce Float64 należało porównać wyniki każdej z nich dla $x = 8^{-1}, 8^{-2}, 8^{-3}, ....$

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

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

## 6.2 Wyniki

| x | f(x) | g(x) |
|---|---|---|
| $8^{-1}$ | 0.0077822185373186414 | 0.0077822185373187065 |
| $8^{-2}$ | 0.00012206286282867573 | 0.00012206286282875901 |
| $8^{-3}$ | $1.9073468138230965 * 10^{-6}$ | $1.907346813826566 * 10^{-6}$ |
| $8^{-4}$ | $2.9802321943606103 * 10^{-8}$ | $2.9802321943606116 * 10^{-8}$ |
| $8^{-5}$ | $4.656612873077393 * 10^{-10}$ | $4.6566128719931904 * 10^{-10}$ |
| $8^{-6}$ | $7.275957614183426 * 10^{-12}$ | $7.275957614156956 * 10^{-12}$ |
| $8^{-7}$ | $1.1368683772161603 * 10^{-13}$ | $1.1368683772160957 * 10^{-13}$ |
| $8^{-8}$ | $1.7763568394002505 * 10^{-15}$ | $1.7763568394002489 * 10^{-15}$ |
| $8^{-9}$ | 0.0 | $2.7755575615628914 * 10^{-17}$ |
| $8^{-10}$ | 0.0 | $4.336808689942018 * 10^{-19}$ |

## 6.3 Wnioski

Wnioskując z otrzymanych wyników, możemy stwierdzić, że dokładniejsze wyniki otrzymujemy obliczając funkcje g. Dzieje się to dlatego, że w funkcji f odejmujemy od siebie dwie bardzo zbliżone do siebie wartości. Wnioskuje to tym, że tracimy najbardziej znaczace bity przez co wyniki nie są do końca wiarygodne.

# 7 Zadanie 7 - Pochodna oraz różnica błędu

## 7.1 Problem

Należało porównać przyblizoną wartość pochodnej funkcji f w punkcie $x_0 = 1$ z jej rzeczywistą wartością. Gdzie $f(x) = \sin x + \cos 3x$.

Błąd do obliczenia:

$$|f'(x_0) - \widetilde{f}'(x_0)|$$

Natomiast przybliżenie jest liczone ze wzoru:

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

Przy obliczeniach należało używać arytmetyki Float64 oraz $h = 2^{-n}$ , gdzie n = 0,1,2,..52

## 7.2 Wyniki

Wyliczony bład z powyższego wzoru dla poszczególnych n oraz pokazanie zmiany h

| n | h | błąd |
|---|---|---|
| 0 | 1.0 | 1.9010469435800585 |
| 1 | 0.5 | 1.753499116243109 |
| 2 | 0.25 | 0.9908448135457593 |
| ⋮ | ⋮ | ⋮ |
| 27 | $7.450580596923828 * 10^{-9}$ | $3.460517827846843 * 10^{-8}$ |
| 28 | $3.725290298461914 * 10^{-9}$ | $4.802855890773117 * 10^{-9}$ |
| 29 | $1.862645149230957 * 10^{-9}$ | $5.480178888461751 * 10^{-8}$ |
| ⋮ | ⋮ | ⋮ |
| 50 | $8.881784197001252 * 10^{-16}$ | 0.11694228168853815 |
| 51 | $4.440892098500626 * 10^{-16}$ | 0.11694228168853815 |
| 52 | $2.220446049250313 * 10^{-16}$ | 0.6169422816885382 |

## 7.3 Wnioski

Z powyższych wyników można wywnioskować, że błąd danego działania największy jest dla dużych h oraz małych. Oznacza to, że w okolicach środka błąd jest najmniejszy.

Najmniejszy błąd wypada dla n=28 tzn dla $h = 2^{-28}$, nie jest to dokładny środek pomiędzy 0 a 52, ponieważ liczby nie są równomiernie rozmieszczone w reprezentacji bitowej.

Natomiast najwiekszy błąd pojawia się przy skrajnych wartościach h, dlatego, że w obliczeniu $\widetilde{f}'(x_0)$, h jest albo pochłaniane bardziej przez $x_0$ albo h pochłania różnicę funkcji obliczoną w liczniku.