

Agata Jasionowska 229726

Laboratorium – Lista 1

1. Zadanie 1

1.1. Macheeps

1.1.1. Opis problemu

Napisanie w języku `Julia` programu wyznaczającego w sposób iteracyjny epsilon maszynowe (czyli najmniejsze takie liczby $macheps > 0$, że $1.0 + macheps > 1.0$) dla wszystkich dostępnych w tym języku typów zmiennopozycyjnych (`Float16`, `Float32` oraz `Float64`).

1.1.2. Opis rozwiązania

W celu znalezienia liczby $macheps$ dla danego typu liczby zmiennopozycyjnej skorzystano z operacji przesunięcia bitowego w prawo zgodnie z podanymi niżej krokami:

1. Zdefiniowanie zmiennej $a = 1.0$ wybranego typu;
2. (W pętli) Dopóki $1.0 + a > 1.0$ przypisanie aktualnej wartości a w pomocniczej zmiennej b oraz wykonanie przesunięcia bitowego w prawo dla a .

Po zakończeniu wykonywania pętli zmienna b będzie przechowywała ostatnią przypisaną wartość większą niż 0.0, czyli poszukiwany $macheps$.

1.1.3. Wyniki

Uzyskano następujące wyniki dla kolejnych typów zmiennopozycyjnych (Tabela 1).

1.1.4. Wnioski

Uzyskano rozwiązania identyczne ze zwracanymi przez funkcje `eps`, co dowodzi prawidłowości przyjętego sposobu rozwiązania problemu.

typ	<i>macheps</i>	<i>eps(typ)</i>	C
Float16	9.77e−4	9.77e−4	9.77e−4
Float32	1.192 092 9e−7	1.192 092 9e−7	1.192 092 895 507 81e−7
Float64	2.220 446 049 250 313e−16	2.220 446 049 250 313e−16	2.220 446 049 250 31e−16

Tabela 1. Wyniki *macheps* wraz z prawidłowymi wartościami oraz danymi z C.

Aby dowiedzieć się, dlaczego to ta liczba spełnia warunek zadania wystarczy dodać ją do wartości 1.0 i przyrzeć się zapisowi bitowemu rozwiązania. W arytmetyce Float32 uzyskano:

0 01111111 000000000000000000000001

Zatem liczba będzie zgodna z zapisem: $2^0 \cdot 1.\text{mantissa}$, gdzie *mantissa* jest najmniejszą możliwą wartością, co jednocześnie koresponduje z precyzją arytmetyki (im mniejszy epsilon maszynowy, tym większa względna precyzja obliczeń).

1.2. ETA

1.2.1. Opis problemu

Napisanie w języku Julia programu wyznaczającego w iteracyjny sposób liczbę *eta* (taką, że $\text{eta} > 0.0$) dla dostępnych w nim typów zmiennopozycyjnych (Float16, Float32 oraz Float64).

1.2.2. Opis rozwiązania

Algorytm wyznaczenia liczby *eta* jest zbliżony do algorytmu obliczającego wartości *macheps*:

Algorithm 1

```

a ← 1.0
while a/2.0 > 0.0 do
  a ← a/2.0
end while

```

Po zakończeniu wykonywania pętli w zmiennej *a* pozostanie poszukiwana wartość *eta*.

1.2.3. Wyniki

Uzyskano następujące wyniki dla kolejnych typów zmiennopozycyjnych (Tabela 2).

1.2.4. Wnioski

W arytmetyce Float32 liczba *eta* zapisana bitowo to:

0 00000000 000000000000000000000001

Czyli jest to najmniejsza dodatnia liczba, którą można zapisać. Wszystkie bity cechy są równe zero, zatem ta liczba jest nieznormalizowana (subnormal).

typ	<i>eta</i>	nextfloat(0.0)
Float16	6.00e−8	6.00e−8
Float32	1.0e−45	1.0e−45
Float64	5.0e−324	5.0e−324

Tabela 2. Wyniki *eta* wraz z prawidłowymi wartościami.

1.3. MAX

1.3.1. Opis problemu

Napisanie w języku **Julia** programu wyznaczającego w iteracyjny sposób liczbę *MAX* dla dostępnych w nim typów zmiennopozycyjnych (**Float16**, **Float32** oraz **Float64**).

1.3.2. Opis rozwiązania

W celu znalezienia wartości *MAX* dla danego typu liczby zmiennopozycyjnej skorzystano z operacji przesunięcia bitowego w prawo oraz z funkcji **isinf(a)** (zwracającej wartość *true*, jeżeli argument jest nieskończonością) zgodnie z podanymi niżej krokami:

1. Zdefiniowanie zmiennej $a = 1.0$ wybranego typu;
2. (W pętli) Dopóki $a \cdot \text{FloatX}(2.0)$, (gdzie $X \in 16, 32, 64$) jest skończone, wykonanie przesunięcia bitowego w prawo dla a ;
3. Wykonanie $a = a \cdot (\text{FloatX}(2.0) - \text{eps}(a))$.

Po wyjściu z pętli w zmiennej a pozostanie żądana wartość *MAX*.

1.3.3. Wyniki

Uzyskano następujące wyniki dla kolejnych typów zmiennopozycyjnych:

typ	<i>a</i>	realmax(typ)	C
Float16	6.55e+4	6.55e+4	6.5504e+4
Float32	3.402 823 5e+38	3.402 823 5e+38	3.402 823 466 385 288 598e+38
Float64	1.797 693 134 862 315 7e+308	1.797 693 134 862 315 7e+308	1.797 693 134 862 315 708e+308

Tabela 3. Wyniki *MAX* wraz z prawidłowymi wartościami.

1.3.4. Wnioski

Uzyskano rozwiązania identyczne ze zwracanymi przez funkcję **realmax()**, co dowodzi prawidłowości przyjętego sposobu rozwiązania problemu. Wartości *MAX* dla kolejnych typów zmiennopozycyjnych są bardzo zbliżone do maksymalnych wartości deklarowanych w dokumentacji języka **C**.

2. Zadanie 2

2.1. Opis problemu

Napisanie w języku `Julia` programu, który eksperymentalnie sprawdzi słuszność stwierdzenia Kahana (epsilon maszynowy może zostać wyznaczony w wyniku obliczenia $3(4/3 - 1) - 1$ w danej arytmetyce zmiennopozycyjnej) dla wszystkich dostępnych typów zmiennopozycyjnych.

2.2. Opis rozwiązania

Obliczenie wartości wyrażenia z użyciem właściwego rzutowania typów zgodnie z poniższym wzorem:

`FloatX(3)·((FloatX(4)/FloatX(3))−FloatX(1))−FloatX(1)`, dla $X \in \{16, 32, 64\}$.

2.3. Wyniki

W wyniku obliczeń otrzymano następujące wyniki dla kolejnych typów zmiennopozycyjnych:

typ	Kachan	<i>macheps</i>
<code>Float16</code>	$-9.77e-4$	$9.77e-4$
<code>Float32</code>	$1.192\,092\,9e-7$	$1.192\,092\,9e-7$
<code>Float64</code>	$-2.220\,446\,049\,250\,313e-16$	$2.220\,446\,049\,250\,313e-16$

Tabela 4. Wyniki twierdzenia Kachana wraz z prawidłowymi wartościami.

2.3.1. Wnioski

Powyższa tabela pokazuje, że prawidłowe rozwiązanie udało się uzyskać jedynie dla typu `Float32`. W dwóch pozostałych przypadkach wynik różnił się bitem znaku. Stwierdzenie Kachana byłoby słuszne, gdyby z wartości wyrażenia wziąć jego wartość bezwzględną.

3. Zadanie 3

3.1. Opis problemu

Napisanie w języku `Julia` programu, który eksperymentalnie sprawdzi, że w arytmetyce `Float64` liczby zmiennopozycyjne są równomiernie rozmieszczone w $[1, 2]$ z krokiem $\delta = 2^{-52}$. Równoznaczne jest to ze stwierdzeniem, iż każda liczby zmiennopozycyjna x z zakresu $[1, 2]$ może zostać przedstawiona jako $x = 1 + k \cdot \delta$ w danej arytmetyce, dla $k = 1, 2, \dots, 2^{52} - 1$ i $\delta = 2^{-52}$.

3.2. Opis rozwiązania

Eksperymentalne sprawdzenie rozmieszczenia liczb zgodne z poniższym schematem:

1. Utworzenie takiej zmiennej δ , że $\delta = 2^k$, $k = 0, -1, -2 \dots$ typu zmiennopozycyjnego;
2. Zdefiniowanie zmiennej a pierwszą wartością przedziału;
3. (W pętli) Zwiększanie a o wartość δ oraz wyświetlenie rezultatu wraz z jego zapisem bitowym (uzyskany przy pomocy funkcji `bits(a)`).

3.3. Wyniki

1. Przedział $[1, 2]$

a	zapis bitowy a
1.0	001111111111000...0001
$1.0 + \delta$	001111111111000...0010
$1.0 + 2 \cdot \delta$	001111111111000...0011
\vdots	\vdots
$2.0 - 3 \cdot \delta$	001111111111111...1101
$2.0 - 2\delta$	001111111111111...1110
$2.0 - \delta$	001111111111111...1111

Tabela 5. Rozmieszczenie liczb w zakresie $[1, 2]$ dla $\delta = 2^{-52}$.

Można zaobserwować, że liczby z zakresu $[1, 2)$ różnią się jedynie mantysą, zaś cecha sprawia, że wyglądają one w następujący sposób: $2^0(=1) \cdot 1.mantissa$. Prowadzi to do wniosku, iż różnica pomiędzy kolejnymi liczbami w tym zakresie jest równa różnicy mantys tych liczb. Zapis bitowy pokazuje, że dodawanie do liczby a wartości $\delta = 2^{-52}$ zwiększa ją o kolejny jeden bit.

2. Przedział $[0.5, 1]$

a	zapis bitowy a
0.5	001111111111000...00010
$0.5 + \delta$	001111111111000...00100
$0.5 + 2 \cdot \delta$	001111111111000...00110
$0.5 + 3 \cdot \delta$	001111111111000...01000
\vdots	\vdots
$1.0 - 3 \cdot \delta$	001111111111111...11011
$1.0 - 2\delta$	001111111111111...11101
$1.0 - \delta$	001111111111111...11111

Tabela 6. Rozmieszczenie liczb w zakresie $[0.5, 1]$ dla $\delta = 2^{-52}$.

Analizę rozmieszczenia liczb rozpoczęto od zbadania zmian w zapisie bitowym dla $\delta = 2^{-52}$. Wyniki widoczne powyżej pokazują cykliczne zwiększanie się a o 2 bity, czyli wartości rozłożone są z dwukrotnie większym krokiem. Zatem rozmieszczenie dla tego przedziału to $\delta = \frac{1}{2} \cdot 2^{-52} = 2^{-53}$.

3. Przedział $[2, 4]$

W ostatnim rozpatrywanym przedziale, z uwagi na dwukrotnie dłuższy

zakres, gęstość liczb jest dwa razy większa. Zatem rozkład liczb w $[2, 4]$ następuje z krokiem $\delta = 2^{-51}$ (Tabela 7).

a	zapis bitowy a
2.0	01000000000000...0000000
$2.0 + \delta$	01000000000000...0000001
$2.0 + 2 \cdot \delta$	01000000000000...0000010
$2.0 + 3 \cdot \delta$	01000000000000...0000011
\vdots	\vdots
$4.0 - 3 \cdot \delta$	01000000000011...1111101
$4.0 - 2\delta$	01000000000011...1111110
$4.0 - \delta$	01000000000011...1111111

Tabela 7. Rozmieszczenie liczb w zakresie $[2, 4]$ dla $\delta = 2^{-51}$.

3.4. Wnioski

Analiza przypadku $[1, 2]$ dowodzi równomiernego rozmieszczenia liczb w tym przedziale z krokiem $\delta = 2^{-52}$, czyli prawdziwy jest wzór: $x = 1 + k \cdot \delta$. Zaobserwowano następujące rozmieszczenie w pozostałych przedziałach:

- $[0.5, 1]$: $x = 1 + k \cdot \delta$, $\delta = 2^{-53}$;
- $[2, 4]$: $x = 1 + k \cdot \delta$, $\delta = 2^{-51}$

Prowadzi to do spostrzeżenia, że im bliżej wartości 0.0 znajdują się liczby, tym gęściej są rozmieszczone.

4. Zadanie 4

4.1. Opis problemu

Napisanie w języku `Julia` programu znajdującego eksperymentalnie taką liczbę zmiennopozycyjną `Float64` $1 < x < 2$, że $x \cdot (1/x) \neq 1$ (tj. $fl(x fl(1/x)) \neq 1$) oraz wyznaczenie najmniejszej takiej wartości.

4.2. Opis rozwiązania

Zastosowanie programu działającego zgodnie z poniższym pseudokodem:

Algorithm 2

```

 $a \leftarrow \text{Float64}(1.0)$ 
while  $a < \text{Float64}(2.0)$  do
    if  $((\text{Float64}(1.0)/a) \cdot a \neq \text{Float64}(1.0))$  then wypisz  $a$ 
    end if
     $a \leftarrow \text{nextfloat}(a)$ 
end while

```

4.3. Wyniki

W wyniku pracy program znaleziono najmniejsze rozwiązanie równe 1.000000057228997. Jest to ciekawy przypadek nieodwracalności dzielenia - próba odwrócenia działania da rozwiązanie : 0.9999999999999999. Taki rezultat wynika z ograniczonej precyzji zapisu i przy wykonywaniu kolejnych operacji może prowadzić do spotęgowania błędów obliczeń.

4.4. Wnioski

Wykonywanie działań na liczbach zmiennopozycyjnych powinno odbywać się zawsze z uwzględnieniem możliwego błędu (wynikłego z zaokrąglania wartości) oraz konsekwencji jego powielania (jak to ma miejsce w przypadku warunku z zadania).

5. Zadanie 5

5.1. Opis problemu

Napisanie w języku Julia implementacji czterech algorytmów obliczających iloczyn skalarny dwóch zadanych wektorów x i y z użyciem typów Float32 oraz Float64.

5.2. Opis rozwiązania

1. „W przód”;
2. „W tył”;
3. Od największego do najmniejszego;

Obliczenie sumy tym algorytmem zostało zaimplementowane w następujący sposób (przykład kodu dla arytmetyki Float32):

```
tab = Float32[]
s1 = Float32(0.0)
s2 = Float32(0.0)
i = 1

while i <= length(x)
    push!(tab, Float32(x[i] * y[i]))
    i += 1
end

sort!(tab, rev=true)
for i in tab
    if(i > 0) s1 += i
end

end

sort!(tab)
for i in tab
```

```

        if(i < 0) s2 += i
        end
end
sum = Float32(s1+s2)

```

4. Od najmniejszego do największego.

Implementacja ostatniego z algorytmów jest analogiczna do kodu z poprzedniego podpunktu. Jedyna różnica polega tutaj na odpowiedniej kolejności sortowania tablicy z sumami częściowymi.

5.3. Wyniki

Poniższa tabela prezentuje uzyskane wyniki dla czterech algorytmów obliczających iloczyn skalarny:

podpunkt	Float32	Float64
1	-0.499 944 3	1.025 188 136 829 667 2e-10
2	-0.454 345 7	-1.564 330 887 049 436 6e-10
3	-0.5	0.0
4	-0.5	0.0

Tabela 8. Obliczanie iloczynu skalarnego wektorów.

5.4. Wnioski

Treść zadania podaje prawidłowy wynik równy $-1.00657107000000e-11$. Wynik iloczynu jest zbliżony do 0.0, co oznacza wektory prostopadłe (ortogonalne), które mają tendencję do generowania dużych błędów względnych.

6. Zadanie 6

6.1. Opis problemu

Obliczenie w języku Julia w arytmetyce Float64 wartości funkcji $f(x) = \sqrt{x^2 + 1} - 1$ oraz $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ dla kolejnych wartości $x = 8^{-1}, 8^{-2}, \dots$

6.2. Opis rozwiązania

Obliczanie wartości obu funkcji w pętli dla kolejnych argumentów.

6.3. Wyniki

Poniżej przedstawiono otrzymane rozwiązania (Tabela 9).

6.4. Wnioski

Analiza uzyskanych rozwiązań pozwala zaobserwować, iż dla argumentu $1 < x < 8$ funkcje dają bardzo zbliżone wartości. Jednak dla $x > 8$ funkcja f

x	$f(x)$	$g(x)$
8^{-1}	$7.7822185373186414e - 3$	$7.7822185373187065e - 3$
8^{-2}	$1.2206286282867573e - 4$	$1.2206286282875901e - 4$
8^{-3}	$1.9073468138230965e - 6$	$1.907346813826566e - 6$
\vdots	\vdots	\vdots
8^{-7}	$1.1368683772161603e - 13$	$1.1368683772160957e - 13$
8^{-8}	$1.7763568394002505e - 15$	$1.7763568394002489e - 15$
8^{-9}	0.0	$2.7755575615628914e - 17$
8^{-10}	0.0	$4.336808689942018e - 19$

Tabela 9. Wartości funkcji $f(x)$ oraz $g(x)$.

zaczyna zwracać 0.0, zaś g podaje dokładny wynik. Pozwala to przypuszczać, że to właśnie jest bardziej wiarygodna funkcja.

Powodem różnic w wynikach obu tych funkcji jest wykonywanie odejmowania na wartościach do siebie zbliżonych. W funkcji f dla bardzo małego x zachodzi wtedy : $\sqrt{x^2 + 1} \approx 1$. Zatem odjęcie liczby bliskiej 1.0 oraz samej 1.0 generuje błąd, który dochodzić może do nawet 100%.

7. Zadanie 7

7.1. Opis problemu

Obliczenie w języku **Julia** w arytmetyce **Float64** przybliżonej wartości pochodnej funkcji $f(x) = \sin x + \cos 3x$ w punkcie $x_0 = 1$ oraz błędów $|f'(x_0) - \tilde{f}(x_0)|$ dla $h = 2^{-n}$ ($n = 0, 1, 2, \dots, 54$).

7.2. Opis rozwiązania

Utworzenie funkcji $f(x)$ oraz jej pochodnej $g(x)$ ($g(x) = \cos x - 3 \sin 3x$). Stworzenie pomocniczych funkcji: obliczającej przybliżoną pochodną oraz błąd pomiaru.

7.3. Wyniki

W wyniku pracy programu uzyskano następujące rezultaty (Tabela 10).

Początkowo błąd maleje wraz ze wzrostem argumentu. Dalej jednak ponownie wzrasta od pewnego n . Spowodowane jest to utratą danych na skutek dodawania do 1.0 bardzo małego $h = 2^{-n}$.

7.4. Wnioski

Zbyt mała wartość h spowoduje utratę dokładności podczas wykonywania operacji $h+1$, co wpłynie na dalsze błędy obliczeń (które mogą być względnie duże).

n	$f'(x)$	błąd
0	2.0179892252685967	1.9010469435800585
1	1.8704413979316472	1.753499116243109
2	1.1077870952342974	0.9908448135457593
\vdots	\vdots	\vdots
16	0.11700383928837255	$6.155759983439424e - 5$
17	0.11697306045971345	$3.077877117529937e - 5$
18	0.11695767106721178	$1.5389378673624776e - 5$
\vdots	\vdots	\vdots
36	0.116943359375	$1.0776864618478044e - 6$
37	0.1169281005859375	$1.4181102600652196e - 5$
38	0.116943359375	$1.0776864618478044e - 6$
\vdots	\vdots	\vdots
52	-0.5	0.6169422816885382
53	0.0	0.11694228168853815
54	0.0	0.11694228168853815

Tabela 10. Wartości funkcji $f(x)$ oraz $g(x)$.