Sprawozdanie 1 - Obliczenia Naukowe

Michał Kallas

26 października 2024

1 Zadanie 1

1.1 Opis problemu

Wyznaczyć iteracyjnie wartości dla typów zmiennopozycyjnych `Float16`, `Float32` oraz `Float64`:

- epsilon maszynowy *macheps*, gdzie $macheps > 0$ to najmniejsza liczba, taka że $fl(1.0 + macheps) > 1.0$ i $fl(1.0 + macheps) = 1.0 + macheps$
- liczba maszynowa *eta*, gdzie $eta > 0$ to najmniejsza liczba możliwa do reprezentacji w danym typie zmiennopozycyjnym
- liczba MAX, gdzie MAX to największa liczba możliwa do reprezentacji w danym typie zmiennopozycyjnym

1.2 Rozwiązanie

Liczby *macheps* oraz *eta* można łatwo wyznaczyć dzieląc jedynkę przez 2, aż do osiągnięcia danego warunku. W przypadku MAX, zamiast tego mnożymy przez 2.

1.3 Wyniki

Typ	Wyliczony *macheps*	Wartość *eps*()	Wartość z pliku `float.h`	Wartość ϵ
Float16	0.000977	0.000977	-	0.0004883
Float32	1.1920929e-7	1.1920929e-7	1.19209290e-7	5.9604645e-8
Float64	2.220446049250313e-16	2.220446049250313e-16	2.2204460492503131e-16	1.1102230246251565e-16

Tabela 1: Porównanie eksperymentalnie wyznaczonego *macheps* z wynikiem funkcji *eps*() z języka Julia, wartościami w pliku nagłówkowym `float.h` z języka C oraz *precyzją arytmetyki*.

Typ	Wyliczona eta	Wartość $nextfloat(0.0)$	Wartość MIN_{sub}
Float16	6.0e-8	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324	5.0e-324

Tabela 2: Porównanie eksperymentalnie wyznaczonego eta z wynikiem funkcji $nextfloat(0.0)$ z języka Julia oraz MIN_{sub}.

Typ	Wyliczony MAX	Wartość $floatmax()$	Wartość z pliku `float.h`
Float16	6.55e4	6.55e4	-
Float32	3.4028235e38	3.4028235e38	3.40282347e38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623157e308

Tabela 3: Porównanie eksperymentalnie wyznaczonego MAX z wynikiem funkcji $floatmax()$ z języka Julia oraz wartościami w pliku nagłówkowym `float.h` z języka C.

Typ	Wartość $floatmin()$	Wartość MIN_{nor}
Float32	1.1754944e-38	1.1754944e-38
Float64	2.2250738585072014e-308	2.2250738585072014e-308

Tabela 4: Porównanie wyników funkcji $floatmin()$ z języka Julia z wartościami MIN_{nor}.

1.4 Obserwacje i wnioski

- Eksperymentalnie wyznaczone $macheps$, eta oraz MAX pokrywają się z wartościami z funkcji bibliotecznych Julii oraz pliku nagłówkowego `float.h` języka C. Wynika to z tego, że oba te języki korzystają ze standardu `IEEE 754` do przedstawiania swoich typów zmiennopozycyjnych
- $macheps = 2 * \epsilon$, gdzie ϵ to *precyzja arytmetyki*
- $eta = MIN_{sub}$, gdzie MIN_{sub} to najmniejsza zdenormalizowana liczba reprezentowana w danym typie
- Wartości zwracane przez funkcję $floatmin()$ są równe MIN_{nor}, gdzie MIN_{nor} to najmniejsza znormalizowana liczba reprezentowana w danym typie

2 Zadanie 2

2.1 Opis problemu

Wyznaczyć eksperymentalnie wartości *macheps* dla typów `Float16`, `Float32`, `Float64` za pomocą wzoru Kahana:

$$macheps = 3(4/3 - 1) - 1$$

Zweryfikować poprawność tego wzoru.

2.2 Wyniki

Typ	Wzór Kahana	*eps*()
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Tabela 5: Porównanie eksperymentalnie wyznaczonego *macheps* ze wzoru Kahana z wynikiem *eps*() z Julii.

2.3 Obserwacje i wnioski

Wyniki ze wzoru Kahana są poprawne po zastosowaniu wartości bezwzględnej. Dla `Float32` wynik jest poprawny, a dla `Float16` i `Float64` ma przeciwny znak. Jest to spowodowane tym, że liczba 4/3 ma rozwinięcie okresowe, co prowadzi do innych ostatnich cyfr mantysy dla różnych typów. Ta ostatnia cyfra jest wykorzystywana do zaokrąglania liczby.

3 Zadanie 3

3.1 Opis problemu

Sprawdzić eksperymentalnie, że w arytmetyce `Float64` liczby są równomiernie rozmieszczone w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$. Sprawdzić jak są rozmieszczone liczby w przedziałach $[\frac{1}{2}, 1]$ i $[2, 4]$.

3.2 Rozwiązanie

Porównywałem wartości powstające na skutek dodania/odjęcia kroku δ z kolejnymi wartościami zwracanymi przez $nextfloat()$ oraz $prevfloat()$. Ze względu na dużą ilość liczb w przedziałach sprawdzałem tylko wartości w okolicy ich początków i końców. Testy wykonałem dla wszystkich 3 przedziałów.

Dla przedziałów $[\frac{1}{2}, 1]$ i $[2, 4]$ krok wyznaczyłem w taki sposób: $\delta = nextfloat(s) - s$, gdzie s to początek przedziału.

3.3 Wyniki

Krok znaleziony dla przedziału $[\frac{1}{2}, 1]$: $\delta = 1.1102230246251565e - 16 = 2^{-53}$

Krok znaleziony dla przedziału $[2, 4]$: $\delta = 4.440892098500626e - 16 = 2^{-51}$

Liczba	*bitstring*(*Liczba*)
1.0	0011111111110000000000000000000000000000000000000000000000000000
$1.0 + \delta$	0011111111110000000000000000000000000000000000000000000000000001
$1.0 + 2\delta$	0011111111110000000000000000000000000000000000000000000000000010
$1.0 + 3\delta$	0011111111110000000000000000000000000000000000000000000000000011
$1.0 + 4\delta$	0011111111110000000000000000000000000000000000000000000000000100

Tabela 6: Porównanie liczb z początku przedziału $[1, 2]$ zwiększanych o $\delta = 2^{-52}$ z ich zapisem binarnym w standardzie `IEEE 754`.

Liczba	*bitstring*(*Liczba*)
0.5	0011111111100000000000000000000000000000000000000000000000000000
$0.5 + \delta$	0011111111100000000000000000000000000000000000000000000000000001
$0.5 + 2\delta$	0011111111100000000000000000000000000000000000000000000000000010
$0.5 + 3\delta$	0011111111100000000000000000000000000000000000000000000000000011
$0.5 + 4\delta$	0011111111100000000000000000000000000000000000000000000000000100

Tabela 7: Porównanie liczb z początku przedziału $[\frac{1}{2}, 1]$ zwiększanych o $\delta = 2^{-53}$ z ich zapisem binarnym w standardzie `IEEE 754`.

Liczba	*bitstring*(*Liczba*)
2.0	0100000000000000000000000000000000000000000000000000000000000000
$2.0 + \delta$	0100000000000000000000000000000000000000000000000000000000000001
$2.0 + 2\delta$	0100000000000000000000000000000000000000000000000000000000000010
$2.0 + 3\delta$	0100000000000000000000000000000000000000000000000000000000000011
$2.0 + 4\delta$	0100000000000000000000000000000000000000000000000000000000000100

Tabela 8: Porównanie liczb z początku przedziału $[2, 4]$ zwiększanych o $\delta = 2^{-51}$ z ich zapisem binarnym w standardzie `IEEE 754`.

3.4 Obserwacje i wnioski

W zadanych przedziałach liczby zmiennopozycyjne są rozmieszczone równomiernie z danymi krokami:

- $\delta = 2^{-52}$ w przedziale $[1, 2]$
- $\delta = 2^{-53}$ w przedziale $[\frac{1}{2}, 1]$

- $\delta = 2^{-51}$ w przedziale $[2, 4]$

Widzimy to po tym, że mantysy w reprezentacji binarnej zwiększają się o jeden, a więc faktycznie nie może istnieć żadna liczba między kolejnymi ich wartościami.

4 Zadanie 4

4.1 Opis problemu

Znaleźć eksperymentalnie w arytmetyce `Float64` najmniejszą liczbę x w przedziale $1 < x < 2$, taką że $x * (1/x) \neq 1$, tj. $fl(xfl(1/x)) \neq 1$.

4.2 Rozwiązanie

Taką liczbę możemy znaleźć sprawdzając wartości kolejnych liczb zmiennopozycyjnych, zaczynając od $x = 1.0$. Kolejne liczby dostarczy nam funkcja $nextfloat()$ z Julii. Sprawdzamy aż do momentu gdy $x * (1/x) \neq 1$.

4.3 Wyniki

Znaleziona liczba to 1.000000057228997.

4.4 Obserwacje i wnioski

W standardzie `IEEE 754` nie mamy gwarancji, że odwrotność liczby będzie poprawna. Nawet przy tak podstawowych operacjach matematycznych na liczbach zmiennopozycyjnych trzeba uważać.

5 Zadanie 5

5.1 Opis problemu

Napisać program obliczający w arytmetyce `Float32` i `Float64` iloczyn skalarny dwóch wektorów:

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

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

Suma powinna być wyliczona na 4 różne sposoby, gdzie $n = 5$:

(a) "w przód", czyli $\sum_{i=1}^{n} x_i y_i$

(b) "w tył", czyli $\sum_{i=n}^{1} x_i y_i$

(c) od największego do najmniejszego, czyli tworząc sumy częściowe dodatnie(dodawane w kolejności malejącej) i ujemne(dodawane w kolejności rosnącej), a następnie je dodając

(d) od najmniejszego do największego, czyli przeciwnie do metody (c)

5.2 Wyniki

Poprawny wynik to $-1.00657107000000 * 10^{-11}$. Poniżej otrzymane wyniki eksperymentalne:

Typ	Metoda a	Metoda b	Metoda c	Metoda d
Float32	-0.4999443	-0.4543457	-0.5	-0.5
Float64	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0

Tabela 9: Porównanie 4 algorytmów do obliczenia iloczynu skalarnego dla wektorów x i y.

5.3 Obserwacje i wnioski

Zarówno dla typu `Float32`, jak i `Float64`, żadna z metod nie pozwoliła uzyskać nam poprawnej wartości. Wyniki jasno pokazują nam, że kolejność sumowania ma znaczenie w arytmetyce zmiennopozycyjnej.

6 Zadanie 6

6.1 Opis problemu

Napisać program obliczający w arytmetyce `Float64` wartość funkcji:

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

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

dla kolejnych wartości argumentu $x = 8^{-1}, 8^{-2}, 8^{-3}, ...$

f i g to matematycznie te same funkcje.

6.2 Wyniki

x	$f(x)$	$g(x)$
8^{-1}	0.0077822185373186414	0.0077822185373187065
8^{-2}	0.00012206286282867573	0.00012206286282875901
8^{-3}	1.9073468138230965e-6	1.907346813826566e-6
8^{-4}	2.9802321943606103e-8	2.9802321943606116e-8
8^{-5}	4.656612873077393e-10	4.6566128719931904e-10
8^{-6}	7.275957614183426e-12	7.275957614156956e-12
8^{-7}	1.1368683772161603e-13	1.1368683772160957e-13
8^{-8}	1.7763568394002505e-15	1.7763568394002489e-15
8^{-9}	0.0	2.7755575615628914e-17
...	...	...
8^{-178}	0.0	1.6e-322
8^{-179}	0.0	0.0

Tabela 10: Porównanie wartości funkcji $f(x)$ i $g(x)$ dla kolejnych ujemnych potęg 8 w arytmetyce `Float64`.

6.3 Obserwacje i wnioski

Mimo tego że $f = g$, funkcje zwracają różne wyniki. Bardziej wiarygodne są wartości $g(x)$. Pomimo początkowego zbliżenia wyników, $f(x)$ osiąga wartość 0 już dla $x = 8^{-9}$, a $g(x)$ dopiero dla $x = 8^{-179}$. Wynika to z tego, że w $f(x)$ odejmujemy bardzo bliskie sobie liczby, jako że $\sqrt{x^2+1} \approx 1$ dla małych wartości x, a odejmujemy od tej liczby 1. Odejmowanie bliskich liczb prowadzi do dużych błędów przez utratę cyfr znaczących. Ten problem nie występuje w $g(x)$, jako że tam nie korzystamy z odejmowania.

7 Zadanie 7

7.1 Opis problemu

Skorzystać ze wzoru na przybliżoną wartość pochodnej:

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

aby wyliczyć w arytmetyce `Float64` wartość dla $f(x) = \sin(x) + \cos(3x)$ w punkcie $x_0 = 1$ oraz błędy $|f'(x_0) - \tilde{f}'(x_0)|$ dla $h = 2^{-n}(n = 0, 1, 2, ..., 54)$.

7.2 Rozwiązanie

W każdej iteracji pętli dla $(n = 0, 1, 2, ..., 54)$ wyznaczałem przybliżoną wartość pochodnej z podanego wzoru oraz dokładną wartość z $f'(x) = \cos(x) - 3\sin(3x)$.

7.3 Wyniki

h	$h+1$	$\tilde{f}'(x_0)$	$\lvert f'(x_0) - \tilde{f}'(x_0)\rvert$
2^{-0}	2.0	2.0179892252685967	1.9010469435800585
2^{-1}	1.5	1.8704413979316472	1.753499116243109
2^{-2}	1.25	1.1077870952342974	0.9908448135457593
...	...	...	...
2^{-27}	1.0000000074505806	0.11694231629371643	3.460517827846843e-8
2^{-28}	1.0000000037252903	0.11694228649139404	4.802855890773117e-9
2^{-29}	1.0000000018626451	0.11694222688674927	5.480178888461751e-8
...	...	...	...
2^{-52}	1.0000000000000002	-0.5	0.6169422816885382
2^{-53}	1.0	0.0	0.11694228168853815
2^{-54}	1.0	0.0	0.11694228168853815

Tabela 11: Porównanie wartości h, $h+1$, przybliżenia pochodnej oraz błędu.

7.4 Obserwacje i wnioski

Mimo tego że w matematyce wartości h bliższe zeru zawsze prowadzą do lepszego przybliżenia, to w arytmetyce zmiennopozycyjnej tak nie jest. Dla typu `Float64` zmniejszanie h poprawia przybliżenie wartości pochodnej aż do $h = 2^{-28}$. Następnie wielkość błędu rośnie. Jest to spowodowane odejmowaniem od siebie zbliżonych liczb, jako że $f(x_0 + h) \approx f(x_0)$ dla małych wartości h.