Sprawozdanie Obliczenia Naukowe
lista 1

Łukasz Bratos

październik 2019

1 Zadanie 1

1.1 O zadaniu

Epsilonem maszynowym *macheps* nazywamy najmniejszą liczbę *macheps* > 0 taką, że $fl(1.0 + macheps) > 1.0$.

Celem pierwszej części zadania jest wyznaczenie iteracyjnie wartości *macheps* dla typów *Float16, Float32, Float64* oraz porównanie ich z wartościami zwracanymi przez funkcję biblioteczną *eps()*. Dodatkowo mamy sprawdzić wartości znajdujące się w pliku nagłówkowym *float.h*.

W drugiej części chcemy wyznaczyć liczbę *eta* dla typów *Float16, Float32, Float64* taką, że *eta* > 0.0 i porównać wyniki z wartościami zwracanymi przez funkcję *nextfloat(TYPE(0.0))*.

W ostatniej części zadania wyznaczamy iteracyjnie liczbę MAX dla typów *Float16, Float32, Float64* i porównujemy wyniki z wartościami zwracanymi przez funkcję *floatmax()*.

1.2 Rozwiązanie

Nasz *macheps* wyliczamy w następujący sposób: zmiennej x przypisujemy wartość jeden dla danego typu (korzystamy z funkcji $one(type)$). Następnie wartość x dzielimy przez dwa i sprawdzamy czy zachodzi równanie $1.0 + x = 1.0$. Jeśli tak to kontynuujemy dzielenie x przez dwa i sprawdzamy dalej. W przeciwnym przypadku zwracamy x jako nasz *macheps*.

Wartość *eta* wyliczamy w podobny sposób. Jedynkę dzielimy przez dwa dopóki jest większa od zera. Otrzymana wartość jest naszą wartością *eta* dla danego typu.

Wartość MAX dla danego typu wyznaczamy analogicznie jak w poprzednich przypadkach, czyli zaczynając od jedynki mnożymy ją przez dwa sprawdzając przed wykonaniem działania czy nie osiągamy wartości inf, ponieważ ostatnie mnożenie dałoby taki wynik. Gdy osiągamy wartość „dwukrotnie mniejszą niż nieskończoność" w standardzie $IEEE754$ tworzymy zmienną pomocniczą y, która jest równa połowie naszej dotychczasowej uzyskanej wartości MAX oznaczonej jako x. Do x dodajemy y i sprawdzamy czy jest różny od nieskończoności. Jeśli tak to y dzielimy przez dwa i kontynuujemy powiększanie x. W przeciwnym przypadku zwracamy dotychczasowa wartość x, który jest naszym MAX.

1.3 Wyniki

Tabela 1: Wyniki dla zadania 1 macheps

Typ	macheps	eps()	float.h
Float16	0.000977	0.000977	b.d.
Float32	1.1920929e-7	1.1920929e-7	1.1920929e-7
Float64	2.220446049250313e-16	2.220446049250313e-16	2.220446049250313e-16

Jak widać udało mi się napisać funkcje, które zwracają wyniki zgodne z wartościami funkcji bibliotecznych.

Wartość epsilonu maszynowego to dwukrotność precyzji arytmetyki, wynoszącej $\epsilon = 2^{-t}$. Liczbę MIN_{sub} wyraża się wzorem $2^{-t-1}2^{c_{min}}$ ($c_{min} = 2^{d-1} + 2$, d i t to ilość bitów przeznaczona odpowiednio na eksponentę i mantysę), co w wyniku daje wartość *eta* dla obu arytmetyk. Funkcja $floatmin()$ dla obu typów zwraca wartość MIN_{nor} równą $2^{c_{min}}$.

Tabela 2: Wyniki dla zadania 1 eta

Typ	eta	nextfloat()
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Tabela 3: Wyniki dla zadania 1 maxfloat

Typ	MAX	floatmax()	float.h
Float16	6.55e4	6.55e4	b.d.
Float32	3.4028235e38	3.4028235e38	3.4028235e38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623157e308

1.4 Wnioski

Liczby rzeczywiste reprezentowane w standardzie $IEEE754$ mają skończoną dokładność.

2 Zadanie 2

2.1 O zadaniu

W tym zadaniu mamy sprawdzić czy obliczając wartość wyrażeni zaproponowanego przez Kahna, tj. $3(4/3-1)-1$ w arytmetyce zmiennopozycyjnej otrzymamy wartość epsilona maszynowego.

2.2 Rozwiązanie

Wymnażając podane wartość przez jedynkę w danym typie zmiennopozycyjnym i wykonując podane działania otrzymujemy wartość wyrażenia $3(4/3-1)-1$.

2.3 Wyniki

Tabela 4: Wyniki dla zadania 2

Typ	Wartość wyrażenia	eps()
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Wyliczając wartość wyrażenia zaproponowanego przez Kahana otrzymujemy wartość epsilona maszynowego z dokładnością do znaku. Wynika to z tego, że długość mantysy dla *Float16* to 10, dla *Float32* - 23, a dla *Float64* - 52. Ze względu na okresowe rozwinięcie ułamka 4/3 w reprezentacji binarnej ostatnią cyfrą mantysy dla *Float32* jest 1, a dla *Float16* i *Float64* - 0, co rzutuje na to po której stronie zera kończymy po operacji odejmowania.

2.4 Wnioski

Kahan ma rację z dokładnością do znaku. Niektóre wyrażenia dające w normalnej arytmetyce zero, w arytmetyce zmiennopozycyjnej mogą dać inne wyniki.

3 Zadanie 3

3.1 O zadaniu

W tym zadaniu mamy sprawdzić czy liczby w standardzie $IEEE754$ są równomiernie rozmieszczone na przedziale $[1, 2]$ oraz z jakim krokiem są rozmieszczone na przedziałach $[\frac{1}{2}, 1]$ i $[2, 4]$.

3.2 Rozwiązanie

Wiemy, że różne mantysy w wartościach granicznych przedziału wykluczają możliwość równomiernego rozmieszczenia. Stąd też pierwszą czynnością jest sprawdzenie równości mantys w postaci binarnej. Jeśli są równe to możemy sprawdzić z jakim odstępem są rozmieszczone obliczając wartość wyrażenia:

$$2^{eksponenta-1023} \cdot 2^{-52}$$

gdzie 1023 to bias dla eksponenty w typie *Float64*, a 52 to liczba bitów znaczących w mantysie.

3.3 Wyniki

Tabela 5: Wyniki dla zadania 3

Przedział	Rozkład
$[0.5, 1]$	1.1102230246251565e-16
$[1, 2]$	2.220446049250313e-16
$[2, 4]$	4.440892098500626e-16

Istotnie liczby w typie *Float64* na przedziale $[1, 2]$ są rozmieszczone z krokiem 2^{-52}. Widzimy też, że na przedziale $[\frac{1}{2}, 1]$ liczby są rozmieszczone z dwukrotnie większym krokiem, a na przedziale $[2, 4]$ dwukrotnie mniejszym niż na $[1, 2]$.

3.4 Wnioski

W $IEEE754$ zależnie od przedziału liczby są reprezentowane z różną dokładnością. Przedziały te zależą od kolejnych potęg dwójki, ponieważ wpływają one na liczbę bitów potrzebnych do zakodowania całkowitej części liczby, kosztem bitów części ułamkowej.

4 Zadanie 4

4.1 O zadaniu

Celem tego zdania jest znalezienie takiej liczby zmiennopozycyjnej x z przedziału $1 < x < 2$ w typie *Float64* spełniającej nierówność $x \cdot \frac{1}{x} \neq 1$. Dodatkowo mamy znaleźć najmniejszą taką liczbę.

4.2 Rozwiązanie

Zaczynając od jedynki i korzystając z funkcji $nextfloat()$ sprawdzamy czy kolejne liczby spełniają nierówność. Zaletą tego rozwiązania jest to, że od razu znajdujemy najmniejszą liczbę z takiego przedziału.

4.3 Wyniki

Liczba, która spełnia naszą nierówność to 1.000000057228997. Jest to najmniejsza taka liczba w przedziale $[1, 2]$.

4.4 Wnioski

Korzystając z arytmetyki zmiennopozycyjnej musimy pamiętać, że nawet dla najprostszych obliczeń możemy dostać niedokładne wyniki.

5 Zadanie 5

5.1 O zadaniu

W zadaniu mamy obliczyć iloczyn skalarny dla dwóch wektorów przy pomocy podanych algorytmów:

- W przód $\Sigma_{i=1}^{n} x_i y_i$

- W tył $\Sigma_{i=n}^{1} x_i y_i$
- Od największego do najmniejszego (dodaj dodatnie liczby w porządku od największego do najmniejszego, dodaj ujemne liczby w porządku od najmniejszego do największego,a następnie daj do siebie obliczone sumy częściowe)
- Od najmniejszego do największego (przeciwnie do poprzedniej metody)

5.2 Rozwiązanie

Rozwiązanie polega na implementacji podanych algorytmów.

5.3 Wyniki

Tabela 6: Wyniki dla zadania 5

Algorytm	Wynik dla Float64	Wynik dla Float32
W przód	1.0251881368296672e-10	-0.3472038161853561
W tył	-1.5643308870494366e-10	-0.3472038162872195
Od największego do najmniejszego	0.0	-0.5
Od najmniejszego do największego	0.0	-0.5

Uzyskane wyniki znacząco różnią się od siebie i co gorsza są różne od prawidłowego wyniku tj. $-1.00657107000000 \cdot 10^{-11}$.

5.4 Wnioski

Kolejność wykonywania dodawań ma wpływ na uzyskane wyniki.

6 Zadanie 6

6.1 O zadaniu

W zadaniu mamy policzyć wartości dwóch równoważnych wyrażeń i porównać wyniki dla argumentów $x = 8^{-1}, 8^{-2}, 8^{-3}, ...$

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

6.2 Rozwiązanie

Wyliczamy explicite wartości podanych wyrażeń.

6.3 Wyniki

Funkcja f znacznie szybciej w porównaniu z g zaczyna zwracać zera. Dla większych argumentów wyniki są całkiem podobne w obu funkcjach, lecz nie są równe. Powodem dla którego $f(x)$ radzi sobie gorzej jest redukcja cyfr znaczących przy odejmowaniu bliskich sobie wartości.

6.4 Wnioski

Należy unikać odejmowania od siebie bliskich wartości, aby nie utracić precyzji.

Tabela 7: Wyniki dla zadania 6

x	$f(8^x)$	$g(8^x)$
1	0.0077822185373186414	0.0077822185373187065
2	0.00012206286282867573	0.00012206286282875901
3	1.9073468138230965e-6	1.907346813826566e-6
4	2.9802321943606103e-8	2.9802321943606116e-8
5	4.656612873077393e-10	4.6566128719931904e-10
6	7.275957614183426e-12	7.275957614156956e-12
7	1.1368683772161603e-13	1.1368683772160957e-13
8	1.7763568394002505e-15	1.7763568394002489e-15
9	0.0	2.7755575615628914e-17
10	0.0	4.336808689942018e-19
20	0.0	3.76158192263132e-37
40	0.0	2.8298997121333476e-73
60	0.0	2.1289799200040754e-109
80	0.0	1.6016664761464807e-145
100	0.0	1.204959932551442e-181
120	0.0	9.065110999561118e-218
140	0.0	6.819831532519088e-254
160	0.0	5.1306710016229703e-290
180	0.0	0.0
200	0.0	0.0

7 Zadanie 7

7.1 O zadaniu

Zadanie polega na sprawdzeniu dokładności liczenia pochodnej funkcji przy pomocy wzoru

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

Testy będziemy wykonywać na funkcji

$$f(x) = sin(x) + cos(3x)$$

Jej dokładna pochodna to

$$f'(x) = cos(x) - 3sin(3x)$$

Testy będziemy wykonywali dla coraz to mniejszych wartości h.

7.2 Rozwiązanie

Rozwiązanie polega na dosłownej implementacji powyższych wzorów.

7.3 Wyniki

Dokładną wartością pochodnej w punkcie $x_0 = 1$ jest 0.11694228168853815. Od tej wartości liczona jest różnica. Co ciekawe najmniejszy błąd jest osiągany dla wartości $h = 2^{-28}$. Dla mniejszych wartości tracimy precyzję.

7.4 Wnioski

Dobrze jest unikać w obliczeniach wartości bardzo blskich zeru, gdyż może to zaważyć na precyzji.

Tabela 8: Wyniki dla zadania 7

h	1 + h	pochodna	różnica
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^{-3}	1.125	0.6232412792975817	0.5062989976090435
2^{-4}	1.0625	0.3704000662035192	0.253457784514981
2^{-5}	1.03125	0.24344307439754687	0.1265007927090087
2^{-6}	1.015625	0.18009756330732785	0.0631552816187897
2^{-7}	1.0078125	0.1484913953710958	0.03154911368255764
2^{-8}	1.00390625	0.1327091142805159	0.015766832591977753
2^{-9}	1.001953125	0.1248236929407085	0.007881411252170345
2^{-10}	1.0009765625	0.12088247681106168	0.0039401951225235265
2^{-11}	1.00048828125	0.11891225046883847	0.001969968780300313
2^{-12}	1.000244140625	0.11792723373901026	0.0009849520504721099
2^{-13}	1.0001220703125	0.11743474961076572	0.0004924679222275685
2^{-14}	1.00006103515625	0.11718851362093119	0.0002462319323930373
2^{-15}	1.000030517578125	0.11706539714577957	0.00012311545724141837
2^{-16}	1.0000152587890625	0.11700383928837255	6.155759983439424e-5
2^{-17}	1.0000076293945312	0.11697306045971345	3.077877117529937e-5
2^{-18}	1.0000038146972656	0.11695767106721178	1.5389378673624776e-5
2^{-19}	1.0000019073486328	0.11694997636368498	7.694675146829866e-6
2^{-20}	1.0000009536743164	0.11694612901192158	3.8473233834324105e-6
2^{-21}	1.0000004768371582	0.1169442052487284	1.9235601902423127e-6
2^{-22}	1.000000238418579	0.11694324295967817	9.612711400208696e-7
2^{-23}	1.0000001192092896	0.11694276239722967	4.807086915192826e-7
2^{-24}	1.0000000596046448	0.11694252118468285	2.394961446938737e-7
2^{-25}	1.0000000298023224	0.116942398250103	1.1656156484463054e-7
2^{-26}	1.0000000149011612	0.11694233864545822	5.6956920069239914e-8
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^{-30}	1.0000000009313226	0.11694216728210449	1.1440643366000813e-7
2^{-31}	1.0000000004656613	0.11694216728210449	1.1440643366000813e-7
2^{-32}	1.0000000002328306	0.11694192886352539	3.5282501276157063e-7
2^{-33}	1.0000000001164153	0.11694145202636719	8.296621709646956e-7
2^{-34}	1.0000000000582077	0.11694145202636719	8.296621709646956e-7
2^{-35}	1.0000000000291038	0.11693954467773438	2.7370108037771956e-6
2^{-36}	1.000000000014552	0.116943359375	1.0776864618478044e-6
2^{-37}	1.000000000007276	0.1169281005859375	1.4181102600652196e-5
2^{-38}	1.000000000003638	0.116943359375	1.0776864618478044e-6
2^{-39}	1.000000000001819	0.11688232421875	5.9957469788152196e-5
2^{-40}	1.0000000000009095	0.1168212890625	0.0001209926260381522
2^{-41}	1.0000000000004547	0.116943359375	1.0776864618478044e-6
2^{-42}	1.0000000000002274	0.11669921875	0.0002430629385381522
2^{-43}	1.0000000000001137	0.1162109375	0.0007313441885381522
2^{-44}	1.0000000000000568	0.1171875	0.0002452183114618478
2^{-45}	1.0000000000000284	0.11328125	0.003661031688538152
2^{-46}	1.0000000000000142	0.109375	0.007567281688538152
2^{-47}	1.000000000000007	0.109375	0.007567281688538152
2^{-48}	1.0000000000000036	0.09375	0.023192281688538152
2^{-49}	1.0000000000000018	0.125	0.008057718311461848
2^{-50}	1.0000000000000009	0.0	0.11694228168853815
2^{-51}	1.0000000000000004	0.0	0.11694228168853815
2^{-52}	1.0000000000000002	-0.5	0.6169422816885382
2^{-53}	1.0	0.0	0.11694228168853815
2^{-54}	1.0	0.0	0.11694228168853815