Wiktoria Byra
nr ind.: 250131
W11, Informatyka, 5 sem.

Obliczenia naukowe – laboratorium 1

Zadanie 1

Problem:
Zadanie polegało na iteracyjnym znalezieniu epsilonu maszynowego, liczby eta oraz górnego zakresu dla typów zmiennopozycyjnych Float16, Float32 i Float64 oraz porównaniu ich z wynikami zwracanymi przez wbudowane funkcje języka Julia.

Rozwiązanie:
W celu obliczenia epsilonu maszynowego ustalamy górny zakres szukanej x na 0.5 i w pętli zmniejszamy zmienną dwukrotnie sprawdzając czy wyrażenie 1 + x jest zaokrąglane do 1. Kiedy trafimy na taką liczbę zwracamy ostatni x, przy którym wyrażenie nie było zaokrąglane do 1 i kończymy działanie pętli.

W celu znalezienia ety, czyli najmniejszej liczby dodatniej w arytmetyce zmiennopozycyjnej ustalamy dolne ograniczenie na 0, a górne na 1. W pętli obliczamy średnią z dolnego i górnego ograniczenia i w zależności od wyniku przesuwamy odpowiednie ograniczenie, co w praktyce również sprowadza się do zmniejszania w pętli dwukrotnie górnego ograniczenia (tak jak przy poszukiwaniu macheps).

W celu znalezienia maksymalnego zakresu danego typu ustalamy zmienną x na najmniejszą liczbę dodatnią. Zwiększamy ją w pętli dwukrotnie dopóki nie będzie oznaczona jako nieskończoność. Wtedy bierzemy ostatnie x, które nie było oznaczone jako nieskończoność i wiemy, że szukane maksimum znajduje się pomiędzy x a 2x. Ostatnim krokiem jest wykonanie algorytmu przypominającego wyszukiwanie binarne: w pętli znajdujemy zmienną będącą średnią dolnego i górnego ograniczenia, jeżeli będzie oznaczona jako nieskończoność, to przesuwamy górne ograniczenie na średnią, w przeciwnym przypadku przesuwamy dolne ograniczenie na średnią. Wykonujemy pętlę dopóki nie znajdziemy zmiennej, która jest oznaczona jako nieskończoność, ale najbliższa kolejna liczba po niej tak. Uwaga: w binsearchu nie możemy przetrzymywać normalnie zapisanego górnego ograniczenia, ponieważ jest ono oznaczone jako nieskończoność i nie moglibyśmy obliczyć średniej, przetrzymujemy więc dolne ograniczenie i odległość od niego do górnego ograniczenia.

Wyniki:

	Float16	Float32	Float64
macheps()	0.000977	1.1920929e-7	2.220446049250313e-16
eps()	0.000977	1.1920929e-7	2.220446049250313e-16
eta()	6.0e-8	1.0e-45	5.0e-324
macheps(type(0.0))	6.0e-8	1.0e-45	5.0e-324
max()	6.55e4	3.4028235e38	1.7976931348623157e308
floatmax()	6.55e4	3.4028235e38	1.7976931348623157e308
Float.h		3.4028234664e+38	1.7976931349e+308

Wyniki napisanych funkcji pokrywają się z wbudowanymi funkcjami języka Julia, od danych z pliku nagłówkowe różnią się precyzją.

Wnioski:
Epsilon maszynowy, czyli Min. Normal jest wartością określającą precyzję arytmetyki na liczbach zmiennoprzecinkowych. Im większa precyzja, tym mniejsza wartość epsilona. Eta, czyli Min. Subnormal jest to najmniejsza dodatnia liczba jaką można zapisać w danym systemie. Różnica pomiędzy macheps a eta: minimum subnormalne (macheps) to najmniejsza liczba w danym systemie, jej cecha wynosi 0, natomiast minimum normalne to najmniejsza mantysa liczby, której cecha wynosi 1. Różne wartości wynikają z różnych gęstości liczb w systemie przy 0 i przy 1.

Zadanie 2

Problem:
Obliczenie epsilonu maszynowego za pomocą wzoru wyznaczonego przez Kahana: 3(4/3-1)-1.

Rozwiązanie:
W celu obliczania epsilonu wykonujemy powyższe działanie w arytmetyce danego typu.

Wyniki:

	Float16	Float32	Float64
kahan()	-0.000977	1.1920929e-7	-2.220446049250313e-16

Wnioski:
Wartości wyników pokrywają się z epsilonem maszynowym obliczonym za pomocą wbudowanej funkcji języka Julia, ale w typach Float16 i Float64 nie zgadza się znak. Wzór byłby poprawny, gdyby nałożyć na niego moduł.

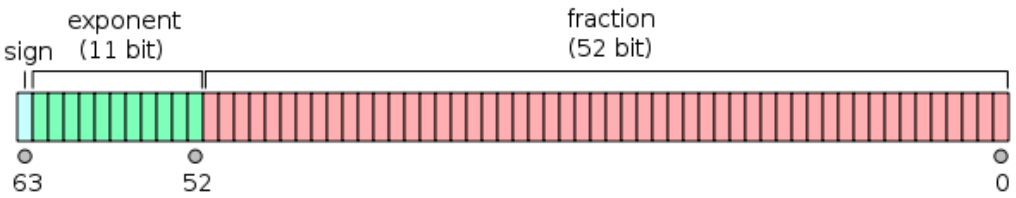
Zadanie 3

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

Rozwiązanie i wyniki:
Program drukuje fragmenty zapisu wszystkich liczb z danych przedziałów w formacie IEEE-754 double floating. Fragmenty, ponieważ interesują nas tylko cyfry od 2 do 12, które odpowiadają za wykładnik liczby 2 w reprezentacji liczby. Wynoszą one:

Dla wszystkich liczb [1,2]: $(01111111111)_2 = (1023)_{10}$
Dla wszystkich liczb $[\frac{1}{2},1]$: $(01111111110\ )_2 = (1022)_{10}$
Dla wszystkich liczb [2,4]: $(10000000000)_2 = (1024)_{10}$

Przyjrzyjmy się bliżej reprezentacji IEEE754 double:



The real value assumed by a given 64-bit double-precision datum with a given biased exponent e and a 52-bit fraction is

$$(-1)^{\text{sign}}(1.b_{51}b_{50}\ldots b_0)_2 \times 2^{e-1023}$$

or

$$(-1)^{\text{sign}}\left(1+\sum_{i=1}^{52} b_{52-i}2^{-i}\right) \times 2^{e-1023}$$

W przedziale [1,2] eksponenta dla wszystkich liczb wynosi 1023, czyli wykładnik wynosi 0, więc ostatnie 52 bity są przepisywane do mantysy bez mnożenia przez 2, może się zmieniać tylko znak przed liczbą. Możemy zatem otrzymać 2^{52} różnych liczb, ponieważ w każdy z 52 bitów można wpisać 0 lub 1. Skoro są rozłożone na przedziale [1,2] to krok wynosi $\delta = \frac{1}{2^{52}} = 2^{-52}$.

W przedziale $[\frac{1}{2},1]$ eksponenta dla wszystkich liczb wynosi 1022, czyli wykładnik wynosi -1, więc ostatnie 52 bity będą przesunięte o jedno miejsce w prawo w mantysie, ponieważ cała liczba w formacie 1,… zostaje przemnożona przez

$2^{-1} = \frac{1}{2}$. Zatem teraz mamy 53 miejsca, w które można wpisać 0 lub 1, ale pierwsza cyfra to zawsze będzie 1, co nadal daje 2^{52} różnych liczb. Skoro liczby są rozmieszczone na przedziale $[\frac{1}{2},1]$ to krok wynosi $\delta = \frac{\frac{1}{2}}{2^{52}} = 2^{-53}$.

W przedziale $[2,4]$ eksponenta dla wszystkich liczb wynosi 1024, czyli wykładnik wynosi 1, więc ostatnie 52 bity będą przesunięte o jedno miejsce w lewo w mantysie, ponieważ cała liczba w formacie 1,… zostaje przemnożona przez $2^1 = 2$. Zatem teraz mamy 51 miejsca po przecinku i jedno przed, w które można wpisać 0 lub 1, co daje 2^{52} różnych liczb. Skoro liczby są rozmieszczone na przedziale $[2,4]$ to krok wynosi $\delta = \frac{2}{2^{52}} = 2^{-51}$.

Zadanie 4

Problem:
Znaleźć w arytmetyce Float64 najmniejszą taką liczbę x w przedziale 1<x<2, że $x * \left(\frac{1}{x}\right) \neq 1$.

Rozwiązanie:
Ustalamy x na najmniejszą liczbę większą od 1. W pętli sprawdzamy czy liczba spełnia równanie, jeżeli nie to zwiększamy ją o macheps. Wykonujemy pętlę dopóki równanie nie zostanie spełnione lub nie dotrzemy do 2.

Wynik:
x = 1.000000057228997

Wnioski:
Taka liczba istnieje, ponieważ nie każdą liczbę zmiennoprzecinkową, będącą wynikiem działania można przedstawić na określonej liczbie bitów, więc operacje na liczbach zmiennoprzecinkowych są obarczone błędem przybliżenia.

Zadanie 5

Problem:
Obliczyć iloczyn skalarny wektorów:
x = [2.718281828, −3.141592654, 1.414213562, 0.5772156649, 0.3010299957]
y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049]
na cztery różne sposoby: sumując iloczyny po kolei, od końca, malejąco i rosnąco.

Rozwiązanie:
Obliczamy wyniki wykorzystując sortowanie tablic i sumy częściowe.

Wyniki:

	Float32	Float64
(a)	-0.4999443	1.0251881368296672e-10
(b)	-0.4543457	-1.5643308870494366e-10
(c)	-0.5	0.0
(d)	-0.5	0.0

Dokładny wynik: $-1.00657107000000 * 10^{-11}$.

Wnioski:
Wyniki iloczynu skalarnego są bliższe dokładnego wyniku w precyzji Float64, ponieważ większa precyzja pozwala na mniejsze zaokrąglenia przy obliczaniu.
Obliczanie sum częściowych (sposoby c i d) jest niekorzystne, ponieważ zwiększa liczbę operacji zaokrąglenia wyniku.

Zadanie 6

Problem:
Obliczyć wartości funkcji:
$f(x) = \sqrt{x^2 + 1} - 1$

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

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

Wyniki:

f(0.125) = 0.0077822185373186414
f(0.015625) = 0.00012206286282867573
f(0.001953125) = 1.9073468138230965e-6
f(0.000244140625) = 2.9802321943606103e-8
f(3.0517578125e-5) = 4.656612873077393e-10
f(3.814697265625e-6) = 7.275957614183426e-12
f(4.76837158203125e-7) = 1.1368683772161603e-13
f(5.960464477539063e-8) = 1.7763568394002505e-15
f(7.450580596923828e-9) = 0.0
f(9.313225746154785e-10) = 0.0

g(0.125) = 0.0077822185373187065
g(0.015625) = 0.00012206286282875901
g(0.001953125) = 1.907346813826566e-6
g(0.000244140625) = 2.9802321943606116e-8
g(3.0517578125e-5) = 4.6566128719931904e-10
g(3.814697265625e-6) = 7.275957614156956e-12
g(4.76837158203125e-7) = 1.1368683772160957e-13
g(5.960464477539063e-8) = 1.7763568394002489e-15
g(7.450580596923828e-9) = 2.7755575615628914e-17
g(9.313225746154785e-10) = 4.336808689942018e-19

Wyniki z dwóch funkcji są podobne, jednak nieco się różnią. Funkcja g(x) daje bardziej wiarygodne wyniki, ponieważ w funkcji f(x) funkcja pod pierwiastkiem dąży do 1 dla x dążącego do 0, a odejmowanie liczb bardzo zbliżonych do siebie daje bardzo małe wyniki, których nie da się zapisać i są zaokrąglane, co powoduje duże błędy. Jeżeli jest taka możliwość, to lepiej jest unikać odejmowania małych liczb i zamieniać je na inne działania.

Zadanie 7

Problem:
Obliczyć przybliżoną wartość pochodnej funkcji $f(x) = sinx + cos3x$ w punkcie $x_0 = 1$oraz błąd przybliżenia dla $h = 2^n$ $(n = 0,1,2, \dots ,54)$.

Rozwiązanie:
Korzystamy ze wzoru na przybliżenie pochodnej:

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

Natomiast do obliczenia dokładnej wartości pochodnej wykorzystujemy wzór:
$f'^{(x_0)} = cosx - 3\sin(3x)$.

Wyniki:

h	błąd $\lvert f'(x_0) - \tilde{f}'(x_0)\rvert$	przybliżenie pochodnej $\tilde{f}'(x_0)$
1.0	1.9010469435800585	2.0179892252685967
0.5	1.753499116243109	1.8704413979316472
0.25	0.9908448135457593	1.1077870952342974
0.125	0.5062989976090435	0.6232412792975817
0.0625	0.253457784514981	0.3704000662035192
0.03125	0.1265007927090087	0.24344307439754687
0.015625	0.0631552816187897	0.18009756330732785
0.0078125	0.03154911368255764	0.1484913953710958
0.00390625	0.015766832591977753	0.1327091142805159
0.001953125	0.007881411252170345	0.1248236929407085
0.0009765625	0.0039401951225235265	0.12088247681106168
0.00048828125	0.001969968780300313	0.11891225046883847
0.000244140625	0.0009849520504721099	0.11792723373901026
0.0001220703125	0.0004924679222275685	0.11743474961076572
6.103515625e-5	0.0002462319323930373	0.11718851362093119
3.0517578125e-5	0.00012311545724141837	0.11706539714577957
1.52587890625e-5	6.155759983439424e-5	0.11700383928837255
7.62939453125e-6	3.077877117529937e-5	0.11697306045971345
3.814697265625e-6	1.5389378673624776e-5	0.11695767106721178

1.9073486328125e-6	7.694675146829866e-6	0.11694997636368498
9.5367431640625e-7	3.8473233834324105e-6	0.11694612901192158
4.76837158203125e-7	1.9235601902423127e-6	0.1169442052487284
2.384185791015625e-7	9.612711400208696e-7	0.11694324295967817
1.1920928955078125e-7	4.807086915192826e-7	0.11694276239722967
5.960464477539063e-8	2.394961446938737e-7	0.11694252118468285
2.9802322387695312e-8	1.1656156484463054e-7	0.116942398250103
1.4901161193847656e-8	5.6956920069239914e-8	0.11694233864545822
7.450580596923828e-9	3.460517827846843e-8	0.11694231629371643
3.725290298461914e-9	4.802855890773117e-9	0.11694228649139404
1.862645149230957e-9	5.480178888461751e-8	0.11694222688674927
9.313225746154785e-10	1.1440643366000813e-7	0.11694216728210449
4.656612873077393e-10	1.1440643366000813e-7	0.11694216728210449
2.3283064365386963e-10	3.5282501276157063e-7	0.11694192886352539
1.1641532182693481e-10	8.296621709646956e-7	0.11694145202636719
5.820766091346741e-11	8.296621709646956e-7	0.11694145202636719
2.9103830456733704e-11	2.7370108037771956e-6	0.11693954467773438
1.4551915228366852e-11	1.0776864618478044e-6	0.116943359375
7.275957614183426e-12	1.4181102600652196e-5	0.1169281005859375
3.637978807091713e-12	1.0776864618478044e-6	0.116943359375
1.8189894035458565e-12	5.9957469788152196e-5	0.11688232421875
9.094947017729282e-13	0.0001209926260381522	0.1168212890625
4.547473508864641e-13	1.0776864618478044e-6	0.116943359375
2.2737367544323206e-13	0.0002430629385381522	0.11669921875
1.1368683772161603e-13	0.0007313441885381522	0.1162109375
5.684341886080802e-14	0.0002452183114618478	0.1171875
2.842170943040401e-14	0.003661031688538152	0.11328125
1.4210854715202004e-14	0.007567281688538152	0.109375
7.105427357601002e-15	0.007567281688538152	0.109375
3.552713678800501e-15	0.023192281688538152	0.09375
1.7763568394002505e-15	0.008057718311461848	0.125
8.881784197001252e-16	0.11694228168853815	0.0
4.440892098500626e-16	0.11694228168853815	0.0
2.220446049250313e-16	0.6169422816885382	-0.5
1.1102230246251565e-16	0.11694228168853815	0.0
5.551115123125783e-17	0.11694228168853815	0.0

Wnioski:

Od pewnego momentu zmniejszanie wartości h nie poprawia przybliżenia, ponieważ przy małym h odjemna i odjemnik w mianowniku wzoru na przybliżenie stają się bardzo bliskimi liczbami, a odejmowanie od siebie bardzo bliskich liczb powoduje duże błędy(patrz zadanie poprzednie).