Obliczenia naukowe

Felix Zieliński 272336

Lista 1

Rozwiązania zadań z 1. listy na przedmiot Obliczenia Naukowe. Programy zostały napisane w języku Julia oraz, gdy było to konieczne, w C.

Zadanie 1.

a. Wyznaczanie iteracyjne epsilonów maszynowych wraz z porównaniem z wartościami zwracanymi przez funkcję esp() oraz z danymi z headera float.h jezyka C.

Iteracyjnie dzielę wartość zmiennej `macheps` przez dwa, zaczynając od wartości 1 w danym typie zmiennoprzecinkowym, aż warunek pętli while `1 + macheps > 1` nie zostanie spełniony.

Typ zmiennoprzecinkowy	Wyznaczona wartośc macheps	eps()	<float.h>
16	0.000977	0.000977	brak
32	1.1920929e-7	1.1920929e-7	1.1920929e-07
64	2.220446049250313e-16	2.220446049250313e-16	2.2204460492503131e-16

Moje wyniki zgadzają się z prawdziwymi wartościami
Wyznaczony epsilon maszynowy pomaga w ustaleniu precyzji zapisu liczb zmiennoprzecinkowych, gdyż jest odległością od 1 do kolejnej liczby możliwej do zaprezentowania w danym typie. Im mniejsza będzie ta wartość, tym większa będzie precyzja względna obliczeń.

b. Wyznaczenie iteracyjnie liczby maszynowej eta wraz z porównaniem z wartościami zwracanymi przez funkcję nextfloat()

Iteracyjnie dzielę wartości zmiennej `eta` przez dwa począwszy od wartości zmiennej równej 1,aż warunek pętli while `eta > 0` nie zostanie spełniony.

Typ zmiennoprzecinkowy	Wyznaczona wartośc eta	nextfloat()
16	6.0e-8	6.0e-8
32	1.0e-45	1.0e-45
64	5.0e-324	5.0e-324

Wartości zwrócone przez

1. `floatmin(Float32)` - 1.1754944e-38

2. `floatmin(Float64)` - 2.2250738585072014e-308

Tutaj również moje wyniki zgadzają się z prawdzimymi wartościami.
Liczba eta odpowiada najmniejszej zdenormalizowanej liczbie dodatniej reprezentowanej w podanej arytmetyce zmiennopozycyjnej. Jest zdenormalizowana, czyli bity cechy mają wartość 0.
Natomiast wartości zwrócone przez `floatmin` są odpowiednikiem tej wartości, ale znormalizowanej.

c. Wyznaczenie iteracyjne liczby MAX wraz z porównaniem z wartościami zwracanymi przez funckje floatmax() oraz z danymi z headera float.h języka C.

Iteracyjnie mnożę wartości zmiennej `max`, aż stanie się ona równa wartości `isinf`. Następnie, w celu poprawienia dokładności obliczeń, dodaję do poprzedniej wartości `max` $\frac{x}{k}$, gdzie `k = 2, 4, ...`, aż `max` będzie równa `isinf` bądź mniejsza od 1

Typ zmiennoprzecinkowy	Wyznaczona wartośc max	floatmax()	<float.h>
16	6.55e4	6.55e4	brak
32	3.4028235e38	3.4028235e38	3.40282347e+38
64	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623157e+308

Moje wyniki zgadzają się z wartościami zwróconymi przez `floatmax()`.
Obie wartości mają postać zdenormalizowaną

Zadanie 2. Sprawdzenie, czy twierdzenie Kahana jest poprawne.

Twierdzenie to mówi, że epsilon maszynowy można uzyskać, obliczając wartość

$$3 * (4/3 - 1) - 1$$

w odpowiedniej arytmetyce zmiennoprzecinkowej. Sprawdzenia dokonuję obliczając tę wartośc dla wartości rzutowanych na podany typ. Jedynkę otrzymuję funkcją `one`.

Typ zmiennoprzecinkowy	Wyznaczona wartość	eps()
16	-0.000977	0.000977
32	1.1920929e-7	1.1920929e-7
64	-2.220446049250313e-1	2.220446049250313e-16

Obliczone wyniki praktycznie pokrywają się z prawdziwymi wartościami - nie licząc znaku. Zmiana znaku dla typów `Float16` oraz `Float64` może wynikać z ilości bitów znaczących w tych typach (odpowiednio, 10 oraz 52). Ponadto, rozwijając 4/3 binarnie, otrzymamy 1.(10). To powoduje, że ostatnią cyfrą mantysy w tych typach będzie 0, co zmienia znak na przeciwny. Tak więc, gdy weźmiemy moduł z obliczonych wartości, otrzymamy poprawne wyniki, więc twierdzenie Kahana w rzeczywistości jest poprawne.

Zadanie 3. Sprawdzenie, czy liczby w arytmetyce `Float(64)` liczby zmiennopozycyjne sa równomiernie rozmieszczone.

Najprostrzym, a zarazem najdłuższym obliczeniowo rozwiązaniem jest iteracja przez wszystkie liczby z danego przedziału w celu porównania z wartościami `nextfloat`. Szybciej jednak jest porównać eksponenty pierwszej oraz ostatniej liczby z przedziału. Jeśli byłyby inne, wykluczyłoby to rownomierny rozkład liczb w tym przedziale.

Funkcja potwierdza, że eksponenty są takie same. Ponadto wypisałem kilka początkowych liczb, co dalej potwierdza równomierne rozmieszczenie:

$$0011111111110000000000000000000000000000000000000000000000000000$$

$$0011111111110000000000000000000000000000000000000000000000000001$$

$$0011111111110000000000000000000000000000000000000000000000000010$$

$$0011111111110000000000000000000000000000000000000000000000000011$$

$$0011111111110000000000000000000000000000000000000000000000000100$$

Wiem, że odchylenie wykładnicze dla Float64 wynosi 1023. Mantysa ma 52 bity znaczące. Z tych informacji mogę wyliczyć, jak rozmieszczone są liczby w danym przedziale. Używam do tego wzoru $2^{\text{exp}-1023} * 2^{-52}$. Wyliczone z tego wzoru kroki prezentują się następująco:

$$[0.5, 1] - 1.1102230246251565e - 16$$

$$[1, 2] - 2.220446049250313e - 16$$

$$[2, 4] - 4.440892098500626e - 16$$

Odległości pomiędzy kolejnymi liczbami rosną wraz ze zwiększaniem się eksponenty, co jest zgodne ze standardem IEEE754, w którym liczby są reprezentowane z dokładnością różną w zależności od przedziału.

Zadanie 4. Znalezienie w arytmetyce `Float(64)` liczbę zmiennopozycyjną `x` w przedziale `1 < x < 2` taką, że $x * (1/x) \neq 1$, a także najmniejszej takiej liczby. Iteracyjnie sprawdzam kolejne wyniki działania $x*(1/x) \neq 1$, iterując po `x` przy użyciu funkcji `nextfloat`. Gdy tylko wynik będzie różny od 1, zwracam go, tym samym otrzymując najmnijeszą wartość w zadanym przedziale.
Najmniejsza znalezione przeze mnie liczba w przedziale (1, 2):

`1.000000057228997`

Niepoprawny wynik działania jest spowodowany niedokładnością, jaką są obarczone działania na liczbach zmiennoprzecinkowych. Trzeba zachować więc ostrożność podczas wykonywania obliczeń i brać pod uwagę, że mogą być one nieidealne.

Zadanie 5. Obliczanie iloczynu skalarnego dwóch wektorów na cztery rózne sposoby.
Zaimplementowałem każdy z podanych w poleceniu sposobów, tak więc funkcja `a` liczy "w przód", od pierwszych indeksów, funkcja `b` "w tył", analogiczne, a `c` oraz `d` liczą, odpowiednio, od największego do najmniejszego oraz od najmniejszego do największego.

Sposób	Float32	Float64	Wartość prawidłowa
1	-0.3472038161853561	1.0251881368296672e-10	-1.00657107000000e-11
2	-0.3472038162872195	-1.5643308870494366e-1	-1.00657107000000e-11
3	-0.3472038162872195	0.0	-1.00657107000000e-11
4	-0.3472038162872195	0.0	-1.00657107000000e-11

Jak widać w tabeli, żaden ze sposobów liczenia nie dał dokładnego wyniku, niezależnie od zastosowanej arytmetyki (`Float32` bądź `Float64`, chociaż ten drugi pozwolił na uzyskanie wyniku bliższego prawdziwej wartości). Widać również, że kolejność wykonywania działań wpływa na wynik.

Zadanie 6. Obliczanie wartości funkcji w arytmetyce `Float64` dla kolejnych wartości argumentu x. Zadane funkcje:

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

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

Powyższe funkcje zaimplementowałem w niezmienionej formie, używając `Float64`.

Wartość 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^{-10}	0.0	4.336808689942018e-19

Jak można zaobserwować, funkcja `f` już przy 8^{-9} jest równa 0. Do tego momentu, obie funkcje mają zbliżone wartości. Funkcja `f` traci dokładność przez operowanie na małych liczbach - odejmowanie od pierwiaska, który w pewnym momencie zostaje przybliżony do 1, wartosci 1. Możnaby temu zapobiec poprzez odpowiednie przekształcenie równania.

Zadanie 7. Obliczenie przybliżonej wartości pochodnej w punkcie x, używając wzoru:

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

Funkcja, której pochodną należało obliczyć, to

$$f(x) = sinx + cos3x$$

w punkcie `x = 1` oraz błędów przybliżeń. Pochodna jest obliczana ze wzoru:

$$f'(x) = cosx - 3sin3x$$

Obliczona dokłada wartość pochodnej to

0.11694228168853815

Zaimplementowałem funkcję, która oblicza wartość funkcji `f(x)` dla zadanej wartości `x`, a także taką liczącą przybliżoną wartość pochodnej oraz błąd. Dokłada wartość pochodnej jest przechowywana w zmiennej globalnej.

Wartość h	Wartość $1+h$	pochodna	błąd
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	0.003940195122523526 5
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

Jak można zauważyć, najlepsza dokładność jest dla `h = 2^{-28}`, gdyż rząd wielkości błędu jest najmniejszy (wynosi `10^{-9}`). Potem błąd zaczyna rosnąć, aż będzie on wielkości dokładnej wartości pochodnej. Dzieje się tak, ponieważ bardzo małe liczby w formacie zmiennoprzecinkowym mają niewystarczającą liczbę cyfr znaczących, co powoduje utratę dokładności podczas obliczeń, zwłaszcza podczas odejmowania bliskich siebie liczb.