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 $2^{\exp-1023} * 2^{-52}$. 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.