Obliczenia Naukowe Sprawozdanie 1

Bohdan Tkachenko

October 2022

Zadanie 1

Epsilon maszynowy (Macheps)

Opis problemu: Wyznacznie epsilonów maszynowych (macheps) dla typów Float16,Float32,Float64. Porównanie otrzymanych wyników z danymi otrzymanymi z bibliotecznej funkcji eps() oraz z danymi zawartymi w pliku nagłówkowym float.h języka C.

Rozwiązanie: Pętla w której zmienna m na początku równa 1 jest dzielona przez 2, aż do momentu gdy $1 + \frac{m}{2} = 0$

Wyniki:

Typ	MyMacheps	Julia eps	float.h
Float16	0.000977	0.000977	0
Float32	1.1920929e-7	1.1920929e-7	1.1920929e-7
Float64	2.220446049250313e-16	2.220446049250313e-16	2.220446049250313e-16

Wnioski: Wyniki macheps wyliczonego iteracyjnie za pomocą prostej pętli pokrywają się z danymi zawartymi w pliku float.h i funkcją eps(). Epsilon maszynowy wiąże się z precyzją arytmetyki. Precyzja arytmetyki jest 2 razy mniejsza niż macheps

Najmniejsza liczba większa od 0 (Eta)

Polecenie: Program, który iteracyjnie wylicza liczbę Eta taką,że Eta > 0

Rozwiązanie: Pętla w której 1 jest dzielona przez 2,aż wynik nie będzie równy 0,zwraca ostatni wynik dzielony przez 2

Wyniki:

Typ	My min	nextfloat	floatmin
Float16	5.960464477539062e-08	5.960464477539062e-08	6.103515625000000e-05
Float32	1.401298464324817e-45	1.401298464324817e-45	1.175494350822288e-38
Float64	4.940656458412465e-324	4.940656458412465e-324	2.225073858507201e-308

Wnioski: Eta -najmniejsza liczba,którą można zapisać w IEEE754.Dodatkowo wszystkie bity cechy są wyzerowane ponieważ jest to liczba zdenormalizowana,natomiast floatmin jest minimalną znormalizowaną liczbą, czyli cecha nie jest wyzerowana

Liczba Max

Polecenie: Napisać program w języku Julia wyznaczający iteracyjnie liczbę maksymalną możliwą do zapisania w danej arytmetyce dla typów Float16,Float32,Float64i porównanie z wartościami floatmax() i przechowywanymi w pliku float.

Typ	My max	Julia max	float.h
Float16	6.550400000000000e+04	6.550400000000000e+04	–
Float32	3.402823466385289e+38	3.402823466385289e+38	3.402823466385289e+38
Float64	1.797693134862316e+308	1.797693134862316e+308	1.797693134862316e+308

Rozwiązanie: Pętla, która mnoży zmienną 1.0 $\times$ 2.0 aż do momentu otrzymania nieskończoności (Funkcja isinf zwróci True). Wtedy wynikiem ostateczny będzie pomnożenie liczby z ostatniej iteracji razy (2.0-macheps())

Wnioski: Wyliczone wartości pokrywają się z floatmax() oraz z implementacją w C, a więc można stwierdzić,że metoda znajdywania max jest poprawna.

Zadanie 2

Polecenie: Sprawdzenie eksperymentalne słuszności twierdzenia Kahana.

Wyniki:

Typ	Julia eps	Khan eps
Float16	0.000977	-0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	2.220446049250313e-16	-2.220446049250313e-16

Wnioski: Można zauważyć, że wartości pokrywają się, zmieniony jest jedynie znak w typach Float16 i Float64. Wiązane jest to z zasadą „round to even", która to w zależności od parzystości mantysy zaokrągla liczbę z niedomiarem lub z nadmiarem (zero na ostatniej pozycji mantysy – zaokrąglenie z niedomiarem, jeden – z nadmiarem). Więc we Float16 i Float64 dostajemy zaokrąglenie z niedomiarem, a w Float32 z nadmiarem

Zadanie 3

Polecenie: Sprawdzenie eksperymentalne w arytmetyce Float64, że liczby są równomiernie rozłożone w przedziale [1,2]. Sprawdzenie zależności dla przedziału$[\frac{1}{2}, 1]$ i [2,4].

Wnioski: Liczby w przedziale [1,2] są równomiernie rozmieszczone, bo gdy do liczby 1, która w reprezentacji Float64 jest przedstawiana jako: 0011111111110000000000000000000000000000000000000000000000000000 dodamy 2^{-52} otrzymujemy: 0011111111110000000000000000000000000000000000000000000000000001

Z tego można wywnioskować, że liczbę zmiennopozycyjną x z przedziału [1,2] można przedstawić za pomocą x = 1 + k δ w tej arytmetyce, gdzie k = 1, 2, . . . , $2^{52} - 1$ i $\delta = 2^{-52}$.

Analogicznie przedstawia się sytuacja dla przedziału $[\frac{1}{2}, 1]$, jednakże 2 podnosimy do potęgi 53, a dla przedziału [2,4] do potęgi 51.

Można zauważyć, że im większy przedział (im wyższe potęgi liczby 2) tym gęstość liczb maleje. W sprawdzanym bowiem przedziałach nie zmienia się cecha, tylko mantysa.

Zadanie 4

Polecenie: Znalezienie najmniejszej liczby x z przedziału (1,2) w Float64 takiej, że $x\frac{1}{x} \neq x$

Rozwiązanie: W celu rozwiązania zadania dla kolejnych liczb x w arytmetyce Float64, zaczynając od najmniejszej liczby większej od 1, zostało sprawdzone czy warunek $x\frac{1}{x} \neq x$ zachodzi Wyniki: najmniejsza taka liczba ma wartość x = 1.000000057228997 a największa ma wartość: x = 1.9999999850988384.

Wnioski: Działania arytmetyczne na liczbach zmiennopozycyjnych mogą generować błędy związane z zaokrąglaniem wyliczonych wartości. Przy używaniu typów zmiennopozycyjnych takie błędy często są nieuniknione, zwłaszcza przy dzieleniu, które nie jest w tej arytmetyce odwracalne.

Zadanie 5

Obliczenie iloczynu skalarnego danych wektorów z wykorzystaniem czterech różnych algorytmów sumowania dla typów Float32 i Float64.

Wyniki:

Typ	alg1	alg2	alg3	alg4
Float32	-0.4999443	-0.4543457	-0.5	-0.5
Float64	1.0251881368296672e-10	-1.5643308870494366e-10	0	0

Table 1: Do zadania 5

Prawidłowy iloczyn skalarny wektorów obliczony bez zaokrąglania danych to $-1.00657107000000 * 10^{11}$. Wszystkie otrzymane wyniki są od niego różne.

Wniosek: Zadanie pokazuje, że kolejność wykonywania działań nie jest bez znaczenia. Na przykład dodanie do bardzo dużej liczby w stosunku do niej bardzo małej generuje błędy, ponieważ mała liczba zostanie w jakimś stopniu zignorowana podczas zaokrąglania wyniku. Jednym ze sposobów na uniknięcie dużych błędów, kiedy inne metody zawodzą, jest użycie arytmetyki o większej precyzji. Użycie Float64 zamiast Float32 w zadaniu w znaczący sposób przybliżyło uzyskane wyniki do poprawnego, jednak nawet to nie dało zadowalających rezultatów.

Zadanie 6

Polecenie: Zadanie polega na obliczeniu kolejnych wartości funkcji, które są tożsame, w arytmetyce Float64. Za argumenty wybieramy kolejną ujemną potęgę liczby osiem.

Wyniki: Można zauważyć, że funkcja $f(x)$ zwraca mniej dokładny wynik, a funkcja $g(x)$ jest bardziej precyzyjna. Dzieje się tak dlatego, że odejmujemy dwie bardzo bliskie sobie liczby, co dzięki poprzedniemu zadaniu zostało udowodnione, że nie otrzymuje się wtedy dokładnego wyniku, czyli precyzja jest bardzo mała, bo występuje redukcja cyfr znaczących Początkowo wartości są do siebie zbliżone. Funkcja f() osiąga wartość zero dla $x = 8^{-9}$, kiedy to g() osiąga wartość 0 istotnie później, dopiero dla $x = 8^{-179}$.

Potęga	$f(x)$	$g(x)$
1	0.0077822185373186414	0.0077822185373187065
2	0.00012206286282867573	0.00012206286282875901
3	1.9073468138230965e-6	1.907346813826566e-6
4	2.9802321943606103e-8	2.9802321943606116e-8
8	1.7763568394002505e-15	1.7763568394002489e-15
9	0	2.7755575615628914e-17
178	0	1.6e-322
179	0	0

Table 2: Do zadania 6

Zadanie 7

Polecenie: Obliczyć przybliżoną wartość $f'(x)$ dla $f(x) = sinx + cos3x$ ze wzoru $f'(x) = \frac{f(x_0+h)-f(x_0)}{h}$ przybliżonej wartości pochodnej funkcji f (x) = sin x + cos 3x w punkcie $x_0 = 1$ oraz błędów $|f'(x_0) - \overline{f'}(x_0)$ dla h^{-n} gdzie n = [1,2....54]

Rozwiązanie : Obliczamy prawdziwą wartość pochodnej w punkcie x_0 ze wzoru $f`(x_0) = cos(x_0) - 3sin(x_0)$

$f'(x) = 0.11694228168853815$

Porównójemy otrzymane ze wzoru przybliżenia z tym wynikiem

Wnioski: Wyniki pokazują,że zmniejszenie h powoduje zmniejszenie blędu, ale tylko do pewnego momentu. Dla $h = 2^{-28}$ otrzymujemy najmniejszy błąd. Jeżeli dalej będziemy zmniejszać h to wtedy

h	h+1	$\overline{f'_h(x_0)}$	$\lvert f'(x_0) - \overline{f'_h(x_0)}\rvert$
2^{-0}	2	2.0179892252685967	1.9010469435800585
2^{-1}	1.5	1.8704413979316472	1.753499116243109
2^{-2}	1.25	1.1077870952342974	0.9908448135457593
...	...	...	...
2^{-28}	0.11694228649139404	4.802855890773117e-9	1.0000000037252903
2^{-29}	0.11694222688674927	5.480178888461751e-8	1.0000000018626451
2^{-30}	0.11694216728210449	1.1440643366000813e-7	1.0000000009313226
...	...	...	...
2^{-52}	-0.5	0.6169422816885382	1.0000000000000002
2^{-53}	0	0.11694228168853815	1.0

Table 3: Do zadania 7

to zaczyna generować coraz większy błąd. Jest to wynik operacji na bliskich sobie liczbach w arytmetyce zmiennopozycyjnej, których precyzja jest bardzo mała. na końcowych iteracjach błąd jest równy pochodnej, ponieważ przybliżenie jest równe zeru