Sprawozdanie

Igor Urbanowicz

Październik 2022

Zadanie 1

Pierwsza cześć zadania polega na znalezieniu epsilonu maszynowego dla **Float16, Float32, Float64** zrobiłem to przez szukanie ostatniej liczby, która spełnia warunek $x + 1 > 1$ w petli while, gdzie x = 2 i w każdej iteracji dzielimy go przez mała liczbe np 1,0001. Warto jest również zapamietać pierwsza liczbe niespełniajaca naszego warunku, aby uzyskać pewien przedział jeżeli nie znajdziemy dokładnej liczby.

Wyniki:
Float16 - 9.765625e-04, bity - 0001000000000001
Float32 - 1.192093e-07, bity - 00110011100000000000000001001001
Float64 - 2.220446e-16, bity - 00111100101000000000000000000000010100100011001011001010001011

W porównaniu z funkcja eps() wydaja sie dwa razy wieksze.

Ta sama metodologia do zera maszynowego tylko warunek końcowy petli to $x > 0$.

Wyniki:
Float16 - 5.960464e-08 , bity - 0000000000000001
Float32 - 1.401298e-45 y, bity - 00000000000000000000000000000001
Float64 - 4.940656e-324, bity - 000000000000000000000000000000000000000000000000000000000

Zgadzaja sie z funkcja nextfloat(FloatXX(0.0)).

Przy użyciu funkcji bitstring() możemy zobaczyć że floatmin() zwraca minimum normatywne w IEEE

Dla MAX używamy tej samej metodologi tylko używamy mnożenia zamiast dzielenia.

Wyniki:

Float16 - 6.55e4, bity - 0111101111111111
Float32 - 3.4021594e38, bity - 01111111011111111111001100110101
Float64 - 1.7976910673505472e308 y = 011111111110111111111111111111011001011010001100 10

Wyniki sa przybliżone z tym co zwraca floatmax(FloatXX).

Zadanie 2

Wyniki dla działania 3(4/31)1 w Julia:
Float16 - -0.000977
Float32 - 1.1920929e-7
Float64 - -2.220446049250313e-16

Wyniki sa dwa razy wieksze od tych które uzyskałem w zadaniu 1 oraz czasami sa ujemne.

Zadanie 3

Przy użyciu komendy nextfloat(1.0) oraz petli while możemy stwierdzić że różnica miedzy różnice miedzy poszczególnymi float'ami na przedziałach [1,2], [1/2, 1], [2,4] sa takie same i wynosza:
[1/2, 1] - 1.1102230246251565e-16
[1, 2] - 2.220446049250313e-16
[2, 4] - 4.440892098500626e-16

Kolejne wartości sa od siebie dwa razy wieksze. Najprawdopodobniej jest to zwiazane z nastepnymi cechami

Zadanie 4

Szukamy liczby która podczas działania $x * (1/x)$ nie daje matematycznie poprawnego wyniku na przedziale $[1, 2]$. Zaimplementowałem program, który dzieli 2.0 na przez liczbe bliska jedynki o raz szuka x niespełniajacego naszych wymagań.
Wyniki:
x1 - 1.99960005999200096660
x(min) - 1.00000064628251106313

Wyniki sa uwarunkowane typem danych jakie wprowadzimy.

Zadanie 5

Zaimplementowałem algorytmy opisane w zadaniu i otrzymałem nastepujace wyniki:
a - 1.0251881368296672e-10
b - -1.5643308870494366e-10
c - 0.0
d - 0.0
Możemy stwierdzić, że wynik jest zależny od kolejności wprowadzonych wyników.

Zadanie 6

Po implementacji algorytmów uzyskaliśmy nastepujace wyniki:
dla $\sqrt{x^2+1} - 1$:
0.00778,
0.00012,
1.90734e-6,
2.98023e-8,
4.65661e-10,
7.27595e-12,
1.13686e-13,
1.77633e-15,
0.0,
0.0,
0.0,

...

dla $x^2/(\sqrt{(}x^2+1)+1)\,:\,0.00778$
0.00012,
1.90735e-6,
2.98023e-8,
4.65661e-10,
7.27595e-12,
1.13686e-13,
1.77635e-15,
2.77555e-17,
4.33680e-19,
6.77623e-21,
...

Mimo tego, że obydwie funkcje sa dokładnie takie same pod wzgledem matematycznym daja inne wyniki.

Zadanie 7

Po implementacji pochodnej widzimy nastepujace wyniki:
0.40418,
0.41912,
0.41351,
0.40741,
0.40356,
0.40144,
0.40033,
0.39976,
...
0.39062,
0.375,
0.375,
0.25,
0.25,
0.5,

0.0,
0.0,
...

W teorii przy zbliżeniu liczb f(x) i f(x+h) nasza pochodna powinna zwiekszać swoja dokładność jednak tak sie nie dzieje. Zerowanie funkcji najprawdopodobniej oznacza, że przybliżenia tych liczb do Float64 stały sie takie same.