Obliczenia Naukowe

Aleksandra Wójcik

Lista 1

1 Zadanie 1

1.1 Wyznaczanie epsilona maszynowego

Epsilon maszynowy w danej arytmetyce to najmniejsza liczba zmiennoprecyzyjna, którą można dodać do 1.0, aby wynik dodawania był równy 1.0.

```
function calculateMachineEpsilon(T::Type)
    machineEpsilon = T(1)

    while T(1) + machineEpsilon > T(1)
        machineEpsilon = machineEpsilon / T(2)
    end

    return machineEpsilon * T(2)
end
```

Wyniki dla różnych arytmetyk:

Arytmetyka	Epsilon maszynowy	Wartość eps()
Float16	0.000977	0.000977
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}
Float64	$2.220446049250313 \times 10^{-16}$	$2.220446049250313 \times 10^{-16}$

Table 1: Wartości epsilon maszynowego i eps() dla różnych arytmetyk.

Przedstawione wyniki wskazują, że funkcja *calculateMachineEpsilon()* pozwala uzyskać dokładne wyniki.

Przyjżyjmy sie teraz precyzji arytmetyki, daną wzorem :

$$\epsilon = \frac{1}{2}\beta^{1-t}$$

gdzie β to baza, a t to długość mantysy. Zatem obliczmy precyzję dla arytmetyk $Float16, Float32, Float64$ i porównajmy ją do uzyskanych epsilonów maszynowych:

Wyznaczone wartości, dowodzą, że epsilon maszynowy jest równe precyzji danej arytmetyki.

Arytmetyka	Epsilon maszynowy	Precyzja arytmetyki
Float16	0.000977	0.000977
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}
Float64	$2.220446049250313 \times 10^{-16}$	$2.220446049250313 \times 10^{-16}$

Table 2: Wartości epsilon maszynowego i eta() dla różnych arytmetyk.

1.2 Wyznaczanie liczby maszynowej eta

Liczba eta to najmniejsza liczba dodatnia w danej arytmetyce zmiennoprzecinkowej.

```
function calculateEta(T::Type)
    eta = T(1)

    while eta / T(2) > T(0)
        eta = eta / T(2)
    end

    return eta
end
```

Wyniki dla różnych arytmetyk:

Arytmetyka	Liczba eta	Wartość nextFloat(0)
Float16	6.0×10^{-8}	6.0×10^{-8}
Float32	1.0×10^{-45}	1.0×10^{-45}
Float64	5.0×10^{-324}	5.0×10^{-324}

Table 3: Wartości liczby eta i nextFloat(0) dla różnych arytmetyk.

Przyjżyjmy się teraz liczbie MIN_{sub}. Jest to najmniejsza liczba w postaci nieznormalizowaniej w danej arytmetyce. Dana jest wzorem

$$MIN_{sub} = 2^{-t+1} * 2^{c_{min}}$$

$$c_{min} = -2^{d-1} + 2$$

Arytmetyka	Liczba eta	MIN_{sub}
Float16	6.0×10^{-8}	6.0×10^{-8}
Float32	1.0×10^{-45}	1.0×10^{-45}
Float64	5.0×10^{-324}	5.0×10^{-324}

Table 4: Wartości liczby eta i MIN_{sub} dla różnych arytmetyk.

Z powyższej tabeli wynika, że MIN_{sub} wartością odpowiada liczbie eta danej arytmetyce.

1.3 Badanie floatmin

Badam wartości zwracane przez floatmin() dla danych arytmatyk.

Arytmetyka	floatmin()
Float16	6.104×10^{-5}
Float32	$1.1754944 \times 10^{-38}$
Float64	$2.2250738585072014 \times 10^{-308}$

Table 5: Wartości zwracane przez funkcje floatmin() dla różnych arytmetyk.

Wartości zwracane przez minfloat() są zauważalnie większe od tych MIN_{sub} dla danych arytmetyk. Przez co można przypuszczać, że funkcja minfloat() zwraca MIN_{nor}.

$$MIN_{nor} = 2^{c_{min}}$$

Arytmetyka	floatmin()	MIN_{nor}
Float16	6.104×10^{-5}	6.104×10^{-5}
Float32	$1.1754944 \times 10^{-38}$	$1.1754944 \times 10^{-38}$
Float64	$2.2250738585072014 \times 10^{-308}$	$2.2250738585072014 \times 10^{-308}$

Table 6: Wartości zwracane przez funkcje floatmin() MIN_{nor} i dla różnych arytmetyk.

Z powyżej przedstawionej tabeli wynika, że rzeczywiście wartości zwracane przez funkcje floatmin() są równoważne z MIN_{nor} dla danej arytmetyki.

1.4 Wyznaczanie MAX dla wszytskich arytmetyk

Zadanie polega na iteracyjnym wyznaczeniu najwiekszej liczby w danej arytmetyce. Rozwiązanie zaczynam od wyznaczenia liczby x=1.0 jako liczby początkowej. Nastepnie mnożę tę liczbę razy 2, aż do momentu uzyskania INF w danej arytmetyce. Finalnie zwracana jest liczba prevfloat(x)

```
function calculateMax(T::Type)
    num = T(1)
    while !isinf(num)
        num = T(2) * num
    end
    return prevfloat(num)
end
```

Wyniki dla różnych arytmetyk:

Arytmetyka	MAX iteracyjnie	Wartość maxfloat()
Float16	6.55×10^4	6.55×10^4
Float32	3.4028235×10^{38}	3.4028235×10^{38}
Float64	$1.7976931348623157 \times 10^{308}$	$1.7976931348623157 \times 10^{308}$

Table 7: Wartości liczby eta i nextFloat() dla różnych arytmetyk.

2 zadanie 2

Badanie wzoru Kahana

Wzór Kahana jest używany do obliczenia epsilon maszynowego. Porównajmy wyniki uzyskane z tego wzoru z obliczonymi wcześniej wartościami epsilon maszynowego.

Arytmetyka	Epsilon maszynowy (obliczony)	Epsilon maszynowy (wzór Kahana)
Float16	0.000977	-0.000977
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}
Float64	$2.220446049250313 \times 10^{-16}$	$-2.220446049250313 \times 10^{-16}$

Table 8: Porównanie obliczonego epsilon maszynowego z wartością uzyskaną z wzoru Kahana.

Z powyżej przedstawionej tabeli wynika, że wartości nie są sobie równe. NAtomiast wartość bezwzględna wartości zwracanej przez wzór Kahana jest równa epsilonowi maszynowemy w danej arytmetyce.

3 Zadanie 3

Arytmetyka Float64 pozwala na zapis liczby zmiennopozycyjnej na 64 bitach. Pierwszy bit odpowiada znakowi liczby, nastepne c =11 pozwala na zapis cechy z przesunięciem (bias = 1023) i pozostałe t= 52 bity służą do zapisu mantysy.

Odległość pomiędzy dwoma sąsiadjącymi liczbami w tej arytmetyce wynosi:

$$2^{-t} \cdot 2^c = 2^{c-t}$$

Z powyższego wyrażenia można zauważyć, że odstepy pomiędzy liczbami nie są równomierne, ponieważ c nie jest wartością stałą. Z tego wynika, że wraz ze zmianą wartości c w zapisie liczb, będzie zmieniać się ich odległość. Dlatego w różnych przedziałach odległości pomiędzy liczbami będą inne.

W przedziale [1,2] cecha wszędzie poza ostatnią liczbą jest równa 0, więc wszytkie odległości pomiędzy liczbami są sobie równe i wynoszą $2^{0-52} = 2^{-52}$. Natomiast odległość pomiędzy ostatnią liczbą z cechą 0 a pierwszą liczbą z cechą 1 również wynosi 2^{-52}.

W przedziale [2,4] cecha wszędzie poza ostatnią liczbą jest równa 1, więc wszytkie odległości pomiędzy liczbami są sobie równe i wynoszą $2^{1-52} = 2^{-51}$.

Natomiast odległość pomiędzy ostatnią liczbą z cechą 0 a pierwszą liczbą z cechą 1 również wynosi 2^{-51}.

W przedziale $[\frac{1}{2},1]$ cecha wszędzie poza ostatnią liczbą jest równa -1, więc wszytkie odległości pomiędzy liczbami są sobie równe i wynoszą $2^{-1-52} = 2^{-53}$. Natomiast odległość pomiędzy ostatnią liczbą z cechą 0 a pierwszą liczbą z cechą 1 również wynosi 2^{-53}.

```
function checkDistance(num::Float64)
    bit_range = 2:12
    binary_str = bitstring(num)[bit_range]
    c = parse(Int, binary_str, base=2)
    exp = c - 1023
    d = 2^Float64(exp) * Float64(2^-52)


    i = num
    for _ in 1:1000000
        nextNum = nextfloat(Float64(i))
        distance = Float64(d)
        if Float64(i + distance) != nextNum
            return false
        end
        i = nextNum
    end
    return true
end
```

To zadanie ukazuje, że liczby w arytmetyce Float64 nie są równomiernie rozmieszczone.Natomiast liczby w pewnych przedziałach są równomiernie rozmieszczone. Można zauważyć, że im dalej liczba znajduje sie od zera maszynowego, tym większą ma odległość między sąsiadami.

4 Zadanie 4

Eksperymentalne wyszukiwanie w arytmetyce Float64 zgodnej ze standardem IEEE 754 najmniejszej liczby zmiennopozycyjnej x w przedziale $1 < x < 2$, takiej, że

$$x(1/x) = 1$$

Algorytm, wyznaczający liczbe x:

```
function experiment()
    epsilon = eps(Float64)
```

```
    num = Float64(1 + epsilon)

    while num != 2
        if Float64(num * (Float64(1) / num)) != Float64(1)
            return num
        end
        num = Float64(num + epsilon)
    end

end
```

Najmniejszą liczba spełniającą powyższe założenia jest **1.000000057228997**

5 Zadanie 5

Eksperymentalne obliczanie iloczynu skalarnego dwoch wektorów.

Algorytm dodający "w przód":

```
function sum1(T::Type)
    sum = T(0)
    for i in 1:n
        sum += T(x[i]) * T(y[i])
    end
    return sum
end
```

Algorytm dodający "w tył":

```
function sum2(T::Type)
    sum = T(0)
    for i in 1:n
        sum += T(x[n-i+1]) * T(y[n-i+1])
    end
    return sum
end
```

Algorytm dodający wartości posortowane od największej do najmniejszej:

```
function sum3(T::Type)
    arr_x, arr_y = T.(x), T.(y)
    arr_s = T.(arr_x .* arr_y)

    positives = sum(sort(filter(n -> n > 0, arr_s), rev=true))
```

```
    negatives = sum(sort(filter(n -> n < 0, arr_s)))
    return positives + negatives

end
```

Algorytm dodający wartości posortowane od najmniejszej do największej: :

```
function sum4(T::Type)
    arr_x, arr_y = T.(x), T.(y)
    arr_s = T.(arr_x .* arr_y)

    positives = sum(sort(filter(n -> n > 0, arr_s)))
    negatives = sum(sort(filter(n -> n < 0, arr_s), rev=true))
    return positives + negatives

end
```

Sposób dodawania	Float32	Float64
W przód	-0.4999443	$1.0251881368296672 \times 10^{-10}$
W tył	-0.4543457	$1.5643308870494366 \times 10^{-10}$
Posortowane od rosnąco	-0.5	0
Posortowane malejąco	-0.5	0

Table 9: Porównanie obliczonego epsilon maszynowego z wartością uzyskaną z wzoru Kahana.

Prawidłowy wynik dodawania wynosi : $1.00657107000000 \times 10^{-11}$. Jak można zauważyć, żaden z uzyskanych wyników nie jest blisko dokładnego wyniku. Na błędny rezultat działania ma wpływ własność owych wektorów. Otóż są one "prawie" prostopadłe. Własność ta wiąze się z wystąpieniem sporych błędów podczas wyznaczania iloczynu skalarnego.

Z zadania wynika, że kolejność wykonywania działań ma duże znaczenie na dokładność wyniku.

6 Zadanie 6

Obliczanie wartości funkcji $f(x)$ $g(x)$.

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

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

x	$f(x)$	$g(x)$
$\frac{1}{8}$	0.0077822185373186414	0.0077822185373187065
$\frac{1}{8^2}$	0.00012206286282867573	0.00012206286282875901
$\frac{1}{8^3}$	$1.9073468138230965 \times 10^{-6}$	$1.907346813826566 \times 10^{-6}$
$\frac{1}{8^4}$	$2.9802321943606103 \times 10^{-8}$	$2.9802321943606116 \times 10^{-8}$
$\frac{1}{8^5}$	$4.656612873077393 \times 10^{-10}$	$4.6566128719931904 \times 10^{-10}$
$\frac{1}{8^6}$	$7.275957614183426 \times 10^{-12}$	$7.275957614156956 \times 10^{-12}$
$\frac{1}{8^7}$	$1.1368683772161603 \times 10^{-13}$	$1.1368683772160957 \times 10^{-13}$
$\frac{1}{8^8}$	$1.7763568394002505 \times 10^{-15}$	$1.7763568394002489 \times 10^{-15}$
$\frac{1}{8^9}$	...	$2.7755575615628914 \times 10^{-17}$

dla kolejnych wartości x=$\frac{1}{8}, \frac{1}{8^2}, \frac{1}{8^3}$...

Obie funkcje zwracają zbliżone do siebie wyniki. Natomiast funkcja $g(x)$ wydaje się byc dokładniejsza. Prawdą jest, źe funkcje są sobie równoważne. Natomiast są one inaczej skonstruowane, co może prowadzić do pewnych różnic. W przypadku funkcji $f(x) = \sqrt{x^2+1} - 1$ dla bardzo małych x, odejmujemy od siebie liczby o bardzo zbliżonej wartości. Działanie to skutkuje utrata liczb znaczących w wyniku zwracanym przez funkcję. Z powodu spadku dokładności dla $x <= \frac{1}{8^9}$ funkca $f(x)$ zwraca 0.0. Zadanie to pokazuję, że odejmowanie liczb o bliskich wartościach powoduje błędy i spadek dokładności. Natomiast przekształcenia matematyczne pozwalają uniknąć owych sytuacji.

7 Zadanie 7

Z podanego w treści zadania wzoru można obliczyć przybliżoną wartość pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x = 1$.

Wykres przedstawiający błędy powstałe przy wyznaczaniu wartości pochodnej

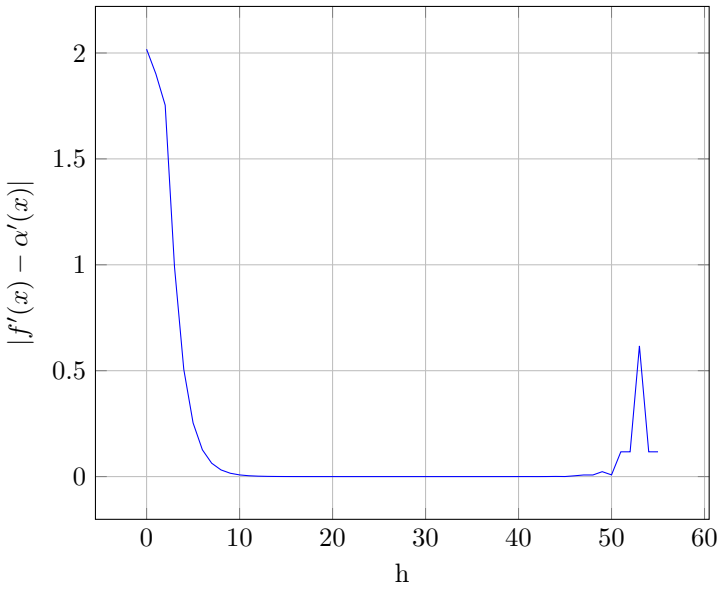

Figure 1: Błędy przy wyznaczaniu wartości pochodnej

Podsumowując, na wykresach przedstawiono błędy obliczeń przy wyznaczaniu wartości pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x = 1$ w zależności od wartości kroku h. W pierwszym wykresie błędy są przedstawione na skali liniowej, a na drugim wykresie zastosowano skalę logarytmiczną. Analizując wykresy, można zauważyć, że zmniejszanie wartości h poprawia dokładność wyników aż do pewnego momentu. Największą dokładność otrzymujemy dla $h = 29$, a dla większych wartości h jakość przybliżeń zaczyna spadać. Wynika to z utraty cyfr znaczących w arytmetyce float64 i podczas odejmowania liczb o zbliżonych wartościach. Wartość $h = 29$ jest optymalna, ponieważ minimalizuje błąd bezwzględny. Dla $h > 29$ błąd rośnie, co jest widoczne na drugim wykresie. Dla bardzo małych wartości h, błąd również rośnie, co wynika z utraty dokładności numerycznej w wyniku zaokrągleń.

Wykres przedstawiający błędy powstałe przy wyznaczaniu wartości pochodnej. Skala logarytmiczna

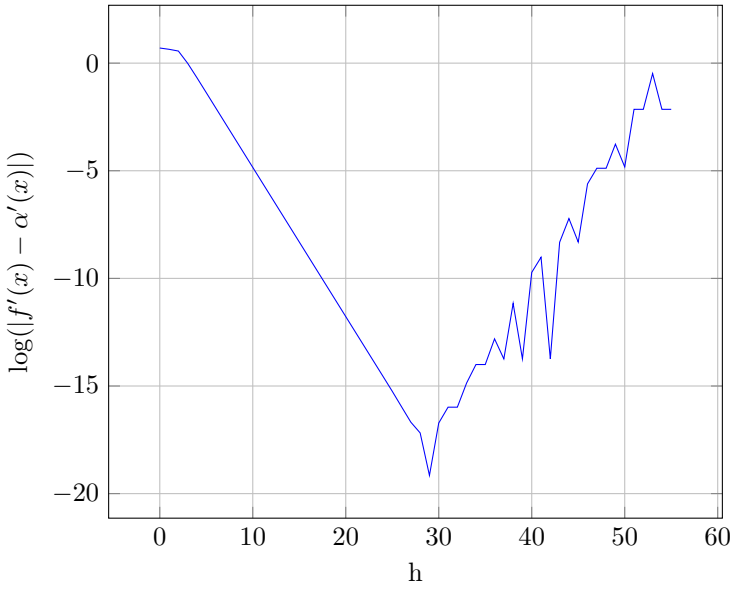

Figure 2: Błędy przy wyznaczaniu wartości pochodnej (skala logarytmiczna)

h	Wartość pochodnej w x	Błąd bezwzględny
1.0	2.0179892252685967	1.9010469435800585
0.5	1.8704413979316472	1.753499116243109
0.25	1.1077870952342974	0.9908448135457593
0.125	0.6232412792975817	0.5062989976090435
0.0625	0.3704000662035192	0.253457784514981
0.03125	0.24344307439754687	0.1265007927090087
0.015625	0.18009756330732785	0.0631552816187897
0.0078125	0.1484913953710958	0.03154911368255764
0.00390625	0.1327091142805159	0.015766832591977753
0.001953125	0.1248236929407085	0.007881411252170345
0.0009765625	0.12088247681106168	0.0039401951225235265
0.00048828125	0.11891225046883847	0.001969968780300313
0.000244140625	0.11792723373901026	0.0009849520504721099
0.0001220703125	0.11743474961076572	0.0004924679222275685
6.103515625e-5	0.11718851362093119	0.0002462319323930373
3.0517578125e-5	0.11706539714577957	0.00012311545724141837
1.52587890625e-5	0.11700383928837255	6.155759983439424e-5
7.62939453125e-6	0.11697306045971345	3.077877117529937e-5
3.814697265625e-6	0.11695767106721178	1.5389378673624776e-5
1.9073486328125e-6	0.11694997636368498	7.694675146829866e-6
9.5367431640625e-7	0.11694612901192158	3.8473233834324105e-6
4.76837158203125e-7	0.1169442052487284	1.9235601902423127e-6
2.384185791015625e-7	0.11694324295967817	9.612711400208696e-7
1.1920928955078125e-7	0.11694276239722967	4.807086915192826e-7
5.960464477539063e-8	0.11694252118468285	2.394961446938737e-7
2.9802322387695312e-8	0.116942398250103	1.1656156484463054e-7
1.4901161193847656e-8	0.11694233864545822	5.6956920069239914e-8
7.450580596923828e-9	0.11694231629371643	3.460517827846843e-8
3.725290298461914e-9	0.11694228649139404	4.802855890773117e-9
1.862645149230957e-9	0.11694222688674927	5.480178888461751e-8
9.313225746154785e-10	0.11694216728210449	1.1440643366000813e-7
4.656612873077393e-10	0.11694216728210449	1.1440643366000813e-7
2.3283064365386963e-10	0.11694192886352539	3.5282501276157063e-7
1.1641532182693481e-10	0.11694145202636719	8.296621709646956e-7
5.820766091346741e-11	0.11694145202636719	8.296621709646956e-7
2.9103830456733704e-11	0.11693954467773438	2.7370108037771956e-6
1.4551915228366852e-11	0.116943359375	1.0776864618478044e-6
7.275957614183426e-12	0.1169281005859375	1.4181102600652196e-5
3.637978807091713e-12	0.116943359375	1.0776864618478044e-6
1.8189894035458565e-12	0.11688232421875	5.9957469788152196e-5
9.094947017729282e-13	0.1168212890625	0.0001209926260381522
4.547473508864641e-13	0.116943359375	1.0776864618478044e-6
2.2737367544323206e-13	0.11669921875	0.0002430629385381522
1.1368683772161603e-13	0.1162109375	0.0007313441885381522
5.684341886080802e-14	0.1171875	0.0002452183114618478
2.842170943040401e-14	0.11328125	0.003661031688538152
1.4210854715202004e-14	0.109375	0.007567281688538152
7.105427357601002e-15	0.109375	0.007567281688538152
3.552713678800501e-15	0.09375	0.023192281688538152
1.7763568394002505e-15	0.125	0.008057718311461848
8.881784197001252e-16	0.0	0.11694228168853815
4.440892098500626e-16	0.0	0.11694228168853815
2.220446049250313e-16	-0.5	0.6169422816885382

Table 10: Wyniki