1 Zadanie 1

Należało wyliczyć iteracyjnie epsilony maszynowe. Epsilony maszynowe wyliczone iteracyjnie, korzystajac z kodu w pliku `macheps.jl`:

- `Float16`: 0.000977
- `Float32`: 1.1920929×10^{-7}
- `Float64`: $2.2204460492540414 \times 10^{-16}$

Te same wartości pojawiaja sie, kiedy użyjemy funkcji `eps()` na tych trzech rodzajach `Float`. W pliku nagłówkowym `float.h` wartość `FLT_EPSILON` jest taka sama, jak `eps(Float32)`, a `DBL_EPSILON` jak `eps(Float64)`.

Należało również wyliczyć maszynowo liczbe eta. Liczba maszynowa eta wyznaczona iteracyjnie, korzystajac z kodu w pliku `eta.jl`:

- `Float16`: 6.0×10^{-8}
- `Float32`: 1.0×10^{-45}
- `Float64`: 5.0×10^{-324}

Wartości `nextfloat(0.0)` dla poszczególnych rodzajów Float odpowiadaja wynikom wyznaczonym iteracyjnie.

$$macheps = \epsilon$$

$$MIN_{sub} < eta = MIN_{norm}$$

Należało również wyliczyć iteracyjnie liczby MAX dla typów zmiennoprzecinkowych. Funkcje `floatmin(Float32)` i `floatmin(Float64)` zwracaja najmniejsza dodatnia normalna liczbe zmiennoprzecinkowa dla odpowiedniego typu - MIN_{norm}. W tabelce iteracyjnie wyznaczone wartości maksymalne wyliczone za pomoca `max.jl`. Takie same wartości możemy znaleźć w pliku `float.h` w C w `FLT_MAX` i `DBL_MAX` oraz zwracane przez funkcje `floatmax()`

- `floatmin(Float32)`: $1.1754944 \times 10^{-38}$
- `floatmin(Float64)`: $2.2250738585072014 \times 10^{-308}$
- MAX dla `Float16`: 6.55×10^{4}
- MAX dla `Float32`: 3.4028235×10^{38}
- MAX dla `Float64`: $1.7976931348623157 \times 10^{308}$

Typ	Wynik	Epsilon maszynowy
Float64	$-2.220446049250313 \times 10^{-16}$	$2.220446049250313 \times 10^{-16}$
Float32	1.1920929×10^{-7}	1.1920929×10^{-7}
Float16	$-3.0000001 \times 10^{32}$	9.7656×10^{-4}

2 Zadanie 2

Tabelka 1 na podstawie obliczeń popełnionych w `kahan.jl`. Dla typu `Float32` algorytm Kahana sprawdził sie idealnie. Dla typu`Float64` z dokładnościa do modułu również uzykaliśmy poprawny epsilon maszynowy. Dla typu `Float16` precyzja okazała sie niewystarczajaca do skutecznego wyliczenia epsilonu maszynowego

3 Zadanie 3

Należy sprawdzić jak rozmieszczone sa liczby w posczególnych przedziałach arytmetyki `Float64`.

Przedział	Rozkład	Krok δ
$[\frac{1}{2}, 1]$	jednostajny	2^-53
$[1, 2]$	jednostajny	2^-52
$[2, 4]$	jednostajny	2^-51

4 Zadanie 4

Należy znaleźć najmniejsza taka liczbe w arytmetyce `Float64`, że $x \cdot 1(1/x) \neq 1$. Liczbe znalazłem za pomoca programu `minfind.jl` i jest to $x = 1.000000057228997$, dla którego zachodzi $x \cdot (1/x) = 0.9999999999999999$

5 Zadanie 5

Należy policzyć iloczyn skalarny na kilka sposobów

Table 1: Porównanie wyników obliczeń iloczynu skalarnego różnymi metodami

Metoda	Iloczyn skalarny	Bład bezwzgledny
Algorytm (a)	$1.0251881368296672 \times 10^{-10}$	$1.1258452438296672 \times 10^{-10}$
Algorytm (b)	$-1.5643308870494366 \times 10^{-10}$	$1.4636737800494365 \times 10^{-10}$
Algorytm (c)	0	$1.0065710700000000 \times 10^{-11}$
Algorytm (d)	0	$1.0065710700000000 \times 10^{-11}$
Wartość dokładna	$-1.0065710700000000 \times 10^{-11}$	—

Eksperyment przeprowadziłem w pliku `vector.jl`. Najlepsza metoda dla tego problemu okazała sie metoda "w przód"

6 Zadanie 6

Pomimo matematycznej równoważności funkcji ich wartości różnia sie dla bardzo małych argumentów $x = 8^{-k}$. Funkcja f(x) daje niewiarygodne wyniki, ponieważ nastepuje utrata precyzji wynikajaca z odjecia dwóch bliskich sobie liczb. Funkcja g(x) jest numerycznie stabilna i dostarcza wiarygodnich wyników nawet dla bardzo małych x. Kod zawarty jest w plku `pierwiastki.jl`

k	x	$f(x)$	$g(x)$
1	1.250000×10^{-1}	$7.782218537318641 \times 10^{-3}$	$7.782218537318706 \times 10^{-3}$
2	1.562500×10^{-2}	$1.220628628286757 \times 10^{-4}$	$1.220628628287590 \times 10^{-4}$
3	1.953125×10^{-3}	$1.907346813823096 \times 10^{-6}$	$1.907346813826566 \times 10^{-6}$
4	$2.44140625 \times 10^{-4}$	$2.980232194360610 \times 10^{-8}$	$2.980232194360612 \times 10^{-8}$
5	$3.0517578125 \times 10^{-5}$	$4.656612873077393 \times 10^{-10}$	$4.656612871993190 \times 10^{-10}$
6	$3.814697265625 \times 10^{-6}$	$7.275957614183426 \times 10^{-12}$	$7.275957614156956 \times 10^{-12}$
7	$4.76837158203125 \times 10^{-7}$	$1.136868377216160 \times 10^{-13}$	$1.136868377216096 \times 10^{-13}$
8	$5.960464477539062 \times 10^{-8}$	$1.776356839400250 \times 10^{-15}$	$1.776356839400249 \times 10^{-15}$
9	$7.450580596923828 \times 10^{-9}$	0.000000000000000	$2.775557561562891 \times 10^{-17}$
10	$9.313225746154785 \times 10^{-10}$	0.000000000000000	$4.336808689942018 \times 10^{-19}$
11	$1.164153218269348 \times 10^{-10}$	0.000000000000000	$6.776263578034403 \times 10^{-21}$
12	$1.455191522836685 \times 10^{-11}$	0.000000000000000	$1.058791184067875 \times 10^{-22}$
13	$1.818989403545856 \times 10^{-12}$	0.000000000000000	$1.654361225106055 \times 10^{-24}$
14	$2.273736754432321 \times 10^{-13}$	0.000000000000000	$2.584939414228211 \times 10^{-26}$
15	$2.842170943040401 \times 10^{-14}$	0.000000000000000	$4.038967834731580 \times 10^{-28}$
16	$3.552713678800501 \times 10^{-15}$	0.000000000000000	$6.310887241768094 \times 10^{-30}$
17	$4.440892098500626 \times 10^{-16}$	0.000000000000000	$9.860761315262648 \times 10^{-32}$
18	$5.551115123125783 \times 10^{-17}$	0.000000000000000	$1.540743955509789 \times 10^{-33}$
19	$6.938893903907228 \times 10^{-18}$	0.000000000000000	$2.407412430484045 \times 10^{-35}$
20	$8.673617379884035 \times 10^{-19}$	0.000000000000000	$3.761581922631320 \times 10^{-37}$

7 Zadanie 7

Należało wykonać eksperymenty na wartości pochodnej funkcji trygonometrycznej. Kod znajduje sie w pliku sinus.jl. Obliczyłem przybliżona wartość pochodnej funkcji $f(x) = sinx + cos3x$ w punkcie $x_0 = 1$ za pomoca wzoru różnicowego dla $h = 2n$ (n od 0 do 54). Poczatkowo zmniejszanie h powoduje zmniejszenie błedu przybliżenia, ale od pewnego momentu (około $h = 2^{20}$) dalsze zmniejszanie h nie poprawia dokładności, a wrecz ja pogarsza. Dzieje sie tak, ponieważ w arytmetyce zmiennoprzecinkowej Float64 dla bardzo małych h wartość $1+h$ jest numerycznie równa 1, co prowadzi do utraty precyzji w obliczeniach różnicowych.

n	h	Przybliżona pochodna	Blad
0	1.00000×10^{0}	2.0179892253	1.90105×10^{0}
1	5.00000×10^{-1}	1.8704413979	1.75350×10^{0}
2	2.50000×10^{-1}	1.1077870952	9.90845×10^{-1}
3	1.25000×10^{-1}	0.6232412793	5.06299×10^{-1}
4	6.25000×10^{-2}	0.3704000662	2.53458×10^{-1}
5	3.12500×10^{-2}	0.2434430744	1.26501×10^{-1}
6	1.56250×10^{-2}	0.1800975633	6.31553×10^{-2}
7	7.81250×10^{-3}	0.1484913954	3.15491×10^{-2}
8	3.90625×10^{-3}	0.1327091143	1.57668×10^{-2}
9	1.95312×10^{-3}	0.1248236929	7.88141×10^{-3}
10	9.76562×10^{-4}	0.1208824768	3.94020×10^{-3}

W efekcie błedy zaokragleń dominuja i przybliżenie pochodnej staje sie mniej dokładne pomimo mniejszego h.