SPRAWOZDANIE NR 1
OBLICZENIA NAUKOWE

Jakub Brodziński
229781
Informatyka WPPT

1 Zadanie 1

1.1 Opis problemu

Zadanie polega na iteracyjnym policzeniu epsilonów maszynowych, liczb eta oraz liczb (MAX) dla wszystkich dostępnych typów zmiennoprzecinkowych **Float16, Float32, Float64**, zgodnych ze standardem IEEE 754 i porównać otrzymane wartości z wartościami przez funkcje przez twórców języka Julia.

1. Epsilon maszynowy to najmniejsza liczba większa od 0, która spełnia równanie:
$$fl(1.0 + macheps) > 1.0$$
2. Eta to najmniejsza liczba, która jest większa od 0.
3. Liczba (MAX) to największa możliwa liczba, którą da się zapisać w standardzie IEEE 754.

1.2 Opis rozwiązania

1. Epsilon maszynowy zostaje policzony poprzez dzielenie liczby $\mathbf{1.0}$ prze liczbę $\mathbf{2.0}$ aż do momentu, gdy jeszcze jedno podzielenie zmiennej sprawiłoby, że nie spełniałaby warunku:
$$fl(1.0 + macheps) > 1.0$$
2. Eta zostaje policzona poprzez dzielenie liczby $\mathbf{1.0}$ przez liczbę $\mathbf{2.0}$ aż do momentu, gdy jeszcze jedno podzielenie zmiennej sprawiłoby, że nie spełniałaby warunku:
$$fl(eta) > 0.0$$
3. Wartość maksymalną otrzymujemy poprzez mnożenie największej liczby, która jest mniejsza niż 1.0 razy 2.0 aż do momentu, gdy jeszcze jedno pomnożenie jej sprawiłoby, że zostałaby zinterpretowana przez Julie jako **„inf”**, czyli nieskończoność. Poprzez zabranie największej liczby mniejszej niż 1.0 jako wynik uzyskamy liczbę, która w mantysie będzie miała same 1.

1.3 Wyniki

W tabeli zostają porównane wyniki otrzymane przeze mnie z wynikami otrzymany poprzez wywołanie funkcji bibliotecznych w Julii. Porównanie następuje poprzez przedstawienie postaci bitowej liczb.

Typ	Obliczony iterycjnie machEps	Właściwy macheps
Float16	0001010000000000	0001010000000000
Float32	00110100000000000000000000000000	00110100000000000000000000000000
Float64	0011110010110000000000000000000000 000000000000000000000000000000	0011110010110000000000000000000000000 000000000000000000000000000

Typ	Obliczona iteracyjnie Eta	Właściwa Eta
Float16	0000000000000001	0000000000000001
Float32	00000000000000000000000000000001	00000000000000000000000000000001
Float64	0000000000000000000000000000000000 000000000000000000000000000001	0000000000000000000000000000000000000 000000000000000000000000001

Typ	Obliczony iteracyjnie MAX	Właściwy MAX
Float16	0111101111111111	0111101111111111
Float32	01111111011111111111111111111111	01111111011111111111111111111111
Float64	0111111111101111111111111111111111 111111111111111111111111111111	0111111111101111111111111111111111111 111111111111111111111111111

1.4 Wnioski

Uzyskane przeze mnie wyniki pokrywają się w pełni z wynikami uzyskanymi przez wywołanie funkcji bibliotecznych Julii.

1. **Epsilon maszynowy**
   Epsilon maszynowy jest dwukrotnie większy niż precyzja arytmetyki **ε**. Przykładowo dla **Float32** epsilon maszynowy wynosi 2^{-52}, natomiast precyzja arytmetyki jest równa 2^{-53} . Wartości dla Float32 (float) oraz Float64(double) w języku C to :
   $$float: 1\quad .1920928955078125e-7$$
   $$double: 2.220446049250313080847263336181640625e-16$$
2. **Eta**
   Poprzez analizę zapisu bitowego **Ety** można wywnioskować, że jest to najmniejsza liczba większa od zera w postaci nieznormalizowanej (MIN_{sub}).
3. **MAX**
   Jest to największa liczba, którą da się przedstawić w danej arytmetyce w standardzie IEE 754. Wszystkie bity mantysy ustawione są na 1. Należy pamiętać, że postać w której wszystkie bity cechy liczby są ustawione na 1 została zarezerwowana dla „specjalnych” przypadków. Wartości dla Float32 (float) oraz Float64 (double) w języku C to :
   $$float: 3\quad .40282346638528859811704183484516925 44e38$$
   $$double: 1\qquad 797693134862315708145274237317043567980e308$$

2 Zadanie 2

2.1 Opis problemu

Celem zadania było sprawdzić czy Kahan słusznie stwierdził, że epsilon mechaniczny można policzyć ze wzoru:

$$3\left(\frac{4}{3}-1\right)-1$$

2.2 Opis rozwiązania

Została zaimplementowana funkcja, która liczy sposobem Kahan’a epsilon mechaniczny dla danego typu danych.

2.3 Wyniki

W tabeli zostały przedstawione postacie bitowe wyników otrzymanych przy użyciu metody wskazanej przez Kahan’a oraz użyciu funkcji z Julii.

Typ	Metoda Kahan’a	Właściwy macheps
Float16	1001010000000000	0001010000000000
Float32	00110100000000000000000000000000	00110100000000000000000000000000

Float64	1011110010110000000000000000000000 000000000000000000000000000000	0011110010110000000000000000000000000 000000000000000000000000000

2.4 Wnioski

Tylko w jednym przypadku wyniki się pokrywają. W dwóch pozostałych wyniki różnią się bitem znaku.

3 Zadanie 3

3.1 Opis problemu

Celem zadania było sprawdzenie czy liczby zmiennopozycyjne są równomiernie rozmieszczone w przedziale $[1,2]$ z krokiem $\delta = 2^{-52}$ oraz czy każdą liczbę można przedstawić w postaci: $x = 1 + k\delta$, gdzie w Float64 $k = 1,2, \dots , 2^{52} - 1$.

Sprawdzić również czy liczby z przedziałów $\left[\frac{1}{2}, 1\right]$ oraz $[2,4]$ zachowują się tak samo.

3.2 Opis rozwiązania

Sprawdzenie czy liczby zmiennopozycyjne w przedziale $[1,2]$ są równomiernie rozmieszczone z danym krokiem przeprowadziłem za pomocą pętli, w której sprawdzałem czy liczba uzyskana przez dodanie do niej $k\delta$ będzie taka sama jak liczba uzyskana przez wywołanie funkcji $nextfloat()$ na zmiennej która przechowywała rezultat z poprzedniej iteracji. Zmienna k była inkrementowana przy każdym wykonaniu pętli.

W przypadku dwóch kolejnych przedziałów również sprawdzałem czy dodawanie do najmniejszej liczby należącej do przedziały δ da ten sam rezultat co wywołanie funkcji $nexfloat()$ na tej samej liczbie.

3.3 Wyniki

Poniższa tabela przedstawia zmienną po dwóch pierwszych przejściach pętli, oraz jak wyglądała gdy $k = 2^{52} - 1$

k	Liczba
0	0011111111110000000000000000000000000000000000000000000000000000
1	0011111111110000000000000000000000000000000000000000000000000001
$2^{52} - 1$	0011111111111111111111111111111111111111111111111111111111111111

Kolejne dwie tabelki przedstawiają wyniki dla przedziałów $\left[\frac{1}{2}, 1\right]$ oraz $[2,4]$.

1/2 + δ	0011111111100000000000000000000000000000000000000000000000000010
nextfloat(Float64(1/2))	0011111111100000000000000000000000000000000000000000000000000001

2.0 + δ	0100000000000000000000000000000000000000000000000000000000000000
nextfloat(Float64(2.0))	0100000000000000000000000000000000000000000000000000000000000001

3.4 Wnioski

Wywołanie funkcji, która zawierała w sobie pętle potwierdziło, że w przedziale [1,2] liczby zmiennopozycyjne są rozmieszczone równomiernie z krokiem, który jest równy δ podanej w treści zadania.

W przypadku dwóch przedziałów $\left[\frac{1}{2}, 1\right]$ oraz $[2{,}4]$ wyniki się różniły z czego można wywnioskować, że w nie we wszystkich przedziałach liczby zmiennopozycyjne są równomiernie rozmieszczone. W przypadku $IEEE$ 754 im przedział jest bliższy zeru tym liczby w nim są gęściej rozmieszczone. W przypadku przedziałów bardziej oddalonych od 0 gęstość rozmieszczenia liczb maleje.

4 Zadanie 4

4.1 Opis problemu

Celem zadania było znalezienie eksperymentalnie w arytmetyce Float64 najmniejszej takiej liczby x większej niż 1 oraz mniejszej niż 2 takiej, że $fl\left(xfl\left(\frac{1}{x}\right)\right) \neq 1$.

4.2 Opis rozwiązania

Liczba została obliczona poprzez wykonanie pętli, która za x brała kolejne liczby zmiennopozycyjne (przy użyciu funkcji $nextfloat()$) i sprawdzała czy warunek, który liczba musiała spełniać zachodzi. Brane pod uwagę było również to, że liczba nie mogła być większe niż 2. Początkową wartością zmiennej było 1.0, tak więc od razu została policzona najmniejsza taka liczba.

4.3 Wyniki

Poniżej została przedstawiona szukana wartość (również binarnie):

$$1.000000057228997$$
$$0011111111110000000000000000000000001111010111001011111100101010$$

4.4 Wnioski

Taka patologia ma miejsce, ze względu na sposób zapisu liczb zmiennoprzecinkowych oraz błędy zaokrągleń. Przy projektowaniu algorytmów lub wykorzystaniu komputera do ważnych obliczeń należy o tym zjawisku pamiętać.

5 Zadanie 5

5.1 Opis problemu

Celem zadania było policzenie było policzenie iloczynu skalarnego 5 wektorów na 4 różne sposoby oraz zwrócenie uwagi jak liczby zmiennopozycyjne zachowują się w działaniach matematycznych. Dokładna wartość sumy wynosi:

$$-1.00657107000000010 * 10^{-11}$$

5.2 Opis rozwiązania

Zostały zaimplementowane 4 różne funkcje, które liczą sumę iloczynów skalarnych dokładnie w ten sposób jaki został przedstawiony w treści zadania.

5.3 Wyniki

Wyniki są przedstawione w poniższej tabeli.

Podpunkt	Wynik w Float32	Wynik w Float64
a)	-0.4999443	$1.0251881368296672e-10$
b)	-0.4543457	$-1.5643308870494366e-10$
c)	-0.5	0.0
d)	-0.5	0.0

5.4 Wnioski

Wyniki działania różnych sposobów obliczania iloczynu skalarnego pokazują, że kolejność wykonywania operacji ma duży wpływ na końcowy wynik. Mimo dobrych wartości sum częściowych końcowe wyniki różnią się od oczekiwanych. Spowodowane jest to redukcją cyfr znaczących.

6 Zadanie 6

6.1 Opis problemu

Celem zadania jest policzenie na dwa różne sposoby tej samej funkcji (z punktu widzenia matematycznego). Wskazać, który sposób liczenia funkcji jest bardziej wiarygodny i dlaczego.

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

6.2 Opis rozwiązania

Funkcje wskazane w treści zadania zostały zaimplementowane. Następnie zostały policzone wartości obu funkcji w punktach wskazanych w treści zadania.

6.3 Wyniki

Wyniki w poniższej tabeli zostały przedstawione dla kilku pierwszych iteracji oraz kilku końcowych

x	$f(x)$	$g(x)$
8^{-1}	0.2806248474865698	1.3902439024390245
8^{-2}	0.00498756211208895	1.00990099009901
8^{-3}	$7.812194848066945e-5$	1.0001562255897516
8^{-7}	$4.6564974098828316e-12$	1.0000000000093132
8^{-8}	$7.260858581048524e-14$	1.0000000000001454
8^{-9}	$1.1102230246251565e-15$	1.0000000000000022
8^{-10}	0.0	1.0
8^{-11}	0.0	1.0
8^{-12}	0.0	1.0

6.4 Wnioski

Mimo tego, że funkcje f oraz g z punktu widzenia matematycznego są sobie równe to wyniki się różnią. Powodem patologia znana jako redukcja cyfr znaczących. W funkcji f ma to miejsce przy odejmowaniu jedynki od wyniku pierwiastka kwadratowego. Dla małych x wartość $\sqrt{x^2+1}$ jest bardzo bliska 1, dlatego też funkcja f jest narażona na redukcje cyfr znaczących. Z tego powodu bardziej wiarygodną funkcją jest funkcja g.

7 Zadanie 7

7.1 Opis problemu

Celem zadania jest policzenie pochodnej funkcji $f(x) = \sin x + \cos 3x$ w punkcie $x_0 = 1$ dla różnych h postaci $h = 2^{-n}$ ($n = 0,1,2,3 \ldots ,54$), na dwa różne sposoby

$$\widetilde{f}'(x_0) = \frac{f(x_0+h) - f(x_0)}{h}$$
$$f'(x_0) = \cos x - 3\sin 3x$$

Należy również policzyć błędy bezwzględne policzonych wartości pochodnych.

7.2 Opis rozwiązania

Zostały zaimplementowane 4 funkcje. Jedna licząca pochodną w punkcie w sposób „przybliżony”, druga ze wzoru na dokładną wartość, trzecia która liczy błąd bezwzględny na wartościach policzonych dwoma sposobami oraz czwarta licząca wartość funkcji f w punkcie.

7.3 Wyniki

Kilka początkowych wyników zostały przedstawione w tabeli poniżej. Prze $\Delta f'$ oznaczyłem błąd bezwzględny.

h	$\widetilde{f}'(x_0)$	$\Delta f'$
2^{0}	2.0179892252685967	1.9010469435800585
2^{-1}	1.8704413979316472	1.753499116243109
2^{-2}	1.1077870952342974	0.9908448135457593
2^{-3}	0.6232412792975817	0.5062989976090435
2^{-4}	0.3704000662035192	0.253457784514981
2^{-5}	0.24344307439754687	0.1265007927090087
2^{-6}	0.18009756330732785	0.0631552816187897
2^{-20}	0.11694612901192158	$3.8473233834324105e-6$
2^{-40}	0.1168212890625	$1.9235601902423127e-6$

7.4 Wnioski

Przez dodawania do 1 bardzo małych h następuje redukcja cyfr znaczących. W takiej sytuacji pojawiają się błędy, a mniejsze wartości h zamiast przybliżać wynik pochodnej do poprawnego wyniku, zaburzają go. Należy zatem dobierać odpowiednie wartości h, aby uniknąć utraty cyfr znaczących.