Obliczenia naukowe

Lista nr 1 (laboratorium)

autor Piotr Szyma

Zadanie 1

1.1 Epsilon maszynowy (macheps)

1.1.1 Treść

Napisać program w języku Julia wyznaczający iteracyjnie epsilony maszynowe dla wszystkich dostępnych typów zmiennopozycyjnych *Float16, Float32, Float64,* zgodnych ze standardem IEEE 754 (half, single, double), i porównać z wartościami zwracanymi przez funkcje: *eps(Float16), eps(Float32), eps(Float64)* oraz z danymi zawartymi w pliku nagłówkowym float.h dowolnej instalacji języka C.

1.1.2 Rozwiązanie

Rozwiązanie zadania polegało na wykonaniu w pętli dzielenia liczby x (początkowo równej 1) przez 2 póki spełniony był warunek $1 + \frac{x}{2} > 1$. Pętla została zbudowana dla trzech typów liczb zmiennoprzecinkowych – *Float16*, *Float32* oraz *Float64*. Wyznaczone przeze mnie liczby porównałem z ich prawdziwymi wartościami dostępnymi dzięki funkcji *eps*.

1.1.3 Wynik

Wyniki dla poszczególnych typów zmiennoprzecinkowych:

	Wyliczone wartości	Wartość funkcji eps
Float16	9.765625000000000e-04	9.765625000000000e-04
Float32	1.192092895507813e-07	1.192092895507813e-07
Float64	2.220446049250313e-16	2.220446049250313e-16

Wszystkie wartości zgadzały się z wartościami zwracanymi przez funkcję *eps*

1.2 ETA

1.2.1 Treść

Napisać program w języku Julia wyznaczający iteracyjnie liczbę *eta* taką, że *eta* > 0.0 dla wszystkich typów zmiennopozycyjnych Float16, Float32, Float64, zgodnych ze standardem IEEE 754 (half, single, double), i porównać z wartościami zwracanymi przez funkcje: *nextfloat(Float16(0.0)), nextfloat(Float32(0.0)), nextfloat(Float64(0.0))*

1.2.2 Rozwiązanie

W tym zadaniu należało znaleźć najmniejszą liczbę x, t. że $x > 0$, tj. jest to najmniejsza liczba rozróżnialna od zera. W celu wyliczenia tej liczby przyjąłem sobie za x = 1 i uruchomiłem pętlę, która dzieliła liczbę x przez 2 póki spełniony był warunek $0.5 \bullet x > 0$. Wyznaczone przeze mnie liczby zestawiłem z wartościami zwracanymi przez funkcję *nextfloat*.

1.2.3 Wynik

Wyniki dla poszczególnych typów zmiennoprzecinkowych:

	Wyliczone wartości	Wartość funkcji nextfloat
Float16	5.960464477539063e-08	5.960464477539063e-08
Float32	1.401298464324817e-45	1.401298464324817e-45
Float64	4.940656458412465e-324	4.940656458412465e-324

W tym wypadku również wyniki wyliczone iteracyjnie potwierdziły się z rzeczywistą wartością zwracaną przez *nextfloat*.

1.3 Maksymalna wartość arytmetyki (MAX)

1.3.1 Treść

Napisać program w języku Julia wyznaczający iteracyjnie liczbę (MAX) dla wszystkich typów zmiennopozycyjnych *Float16, Float32, Float64*, zgodnych ze standardem IEEE 754 (half, single, double), i porównać z wartościami zwracanymi przez funkcje: *realmax(Float16), realmax(Float32), realmax(Float64)* oraz z danymi zawartymi w pliku nagłówkowym float.h dowolnej instalacji języka C lub z danymi z wykładu lub zob. raport.

1.3.2 Rozwiązanie

W celu wyznaczenia MAX dla w/w typów stworzyłem zmienną x, która początkowo była równa 1. Utworzyłem pętlę, w której wykonywałem operację mnożenia x przez 2. Warunkiem końca pętli był moment, w którym funkcja *isinf*, które jako parametr podano $2 \cdot x$ zwracała prawdę. Ponad to, aby znaleźć największą, nie nieskończoną wartość, liczbę x mnożyłem przez $(2 - eps)$, gdzie *eps* zostało wyliczone w podpunkcie 1.1.

1.3.3 Wynik

Wyniki dla poszczególnych typów zmiennoprzecinkowych:

	Wyliczone wartości	Wartość funkcji realmax()
Float16	6.550400000000000e+04	6.550400000000000e+04
Float32	3.402823466385289e+38	3.402823466385289e+38
Float64	1.797693134862316e+308	1.797693134862316e+308

W tym wypadku również liczby zwrócone w wyniku wykonania programu są równe wartościom funkcji *realmax()*.

Jaki jest związek liczby macheps obliczonej przez nas w zadaniu z pojęciem precyzji arytmetyki podanym na wykładzie? Zgodnie z definicją, precyzją arytmetyki nazywamy maksymalny co do wartości błąd względny, jaki może zajść podczas procesu zaokrąglania liczby rzeczywistej do jej odpowiednika w arytmetyce zmiennoprzecinkowej. Macheps równa się więc precyzji arytmetyki.

Jeśli chodzi o relację między MIN_{sub}, a obliczoną przez nas wartością ETA, to – zgodnie z raportem Williama Kahna dotyczącym standardu IEEE 754 – jest to to samo. Oto fragment raportu:

Subnormals, which permit Underflow to be Gradual, are nonzero numbers with an unnormalized significand n and the same minimal exponent k as is used for 0.

W celu porównania wyników z biblioteką float.h języka C, napisałem krótki program importujący bibliotekę float.h oraz wyświetlający w terminalu wartości stałych FLT_EPSILON, DBL_EPSILON, FLT_MAX oraz DBL_MAX, które – odpowiednio – równają się wyliczonym wartościom eps oraz MAX dla Float32 (FLT) oraz Float64 (DBL).

Zadanie 2

2.1 Liczba Kahan'a

2.1.1 Treść

Kahan stwierdził, że epsilon maszynowy (*macheps*) można otrzymać obliczając wyrażenie $3\left(\frac{4}{3}-1\right)-1$ w arytmetyce zmiennopozycyjnej. Sprawdzić eksperymentalnie w języku Julia słuszność tego stwierdzenia dla wszystkich typów zmiennopozycyjnych *Float16, Float32, Float64*.

2.1.2 Rozwiązanie

Wykonałem kod obliczający wyrażenie zaproponowane przez Kahan'a dla poszczególnych arytmetyk. Wyliczone wartości zestawiłem w tabeli wraz z rzeczywistym epsilonem.

2.1.3 Wynik

	Wyliczone wartości	Wartość funkcji eps()
Float16	-9.765625000000000e-04	9.765625000000000e-04
Float32	1.192092895507813e-07	1.192092895507813e-07
Float64	-2.220446049250313e-16	2.220446049250313e-16

Oczywiście z matematycznego punktu widzenia wartość tego wyrażenia to 0, niestety jednak komputer, w związku z ograniczoną precyzją, musi zaokrąglić wartość ułamka $\frac{4}{3}$ do najbliższej mu reprezentacji w arytmetyce zmiennoprzecinkowej, generując pewien względny błąd. Jak widać w powyższej tabeli, wartość obliczona w arytmetyce zmiennoprzecinkowej, co do modułu, jest równy wartościom *macheps* dla poszczególnych arytmetyk – co potwierdza stwierdzenie W. Kahana.

Zadanie 3

3.1 Równomierne rozmieszczenie

3.1.1 Treść

Sprawdź eksperymentalnie w języku Julia, że w arytmetyce Float64 (arytmetyce double w standarcie IEEE 754) liczby zmiennopozycyjne są równomiernie rozmieszczone w [1, 2] z krokiem $\delta = 2^{-52}$. Innymi słowy, każda liczba zmiennopozycyjna x pomiędzy 1 i 2 może być przedstawione następująco $x = 1 + k \cdot \delta$ w tej arytmetyce, gdzie $k = 1, 2, \ldots, 2^{52} - 1$ i $\delta = 2^{-52}$. Jak rozmieszczone są liczby zmiennopozycyjne w przedziale [½, 1], jak w przedziale [2, 4] i jak mogą być przedstawione dla rozpatrywanego przedziału

3.1.2 Rozwiązanie

Eksperymentalnie sprawdziłem krok dla krańców w/w przedziałów. W celu zaobserwowania różnicy użyłem wbudowanej w Julię funkcji *bits*. Okazało się, że współczynnik k dla różnych zakresów różni się. Jest to odpowiednio k = 1 dla [1, 2], k = ½ dla [½, 1] oraz k = 2 dla [2, 4]. Fragment wydruku konsoli przedstawiłem poniżej.

3.1.3 Wynik

Poniżej przedstawiłem tabelę dla poszczególnych zakresów z bitami liczb z lewego krańca każdego z przedziałów oraz ze "skokiem", tj. po dodaniu odpowiednich wartości δ.

Dla x = 1.0, $\delta = 2^{-52}$

liczba	bity
x	0**01111111111**0000000000000000000000000000000000000000000000000000
x + δ	0**01111111111**0000000000000000000000000000000000000000000000000001
nextfloat(x)	0**01111111111**0000000000000000000000000000000000000000000000000001

Dla x = 0.5, $\delta = 2^{-53}$

liczba	bity
x	0**01111111110**0000000000000000000000000000000000000000000000000000
x + δ	0**01111111110**0000000000000000000000000000000000000000000000000001
nextfloat(x)	0**01111111110**0000000000000000000000000000000000000000000000000001

Dla x = 2.0, $\delta = 2^{-51}$

liczba	bity
x	0**10000000000**0000000000000000000000000000000000000000000000000000
x + δ	0**10000000000**0000000000000000000000000000000000000000000000000001
nextfloat(x)	0**10000000000**0000000000000000000000000000000000000000000000000001

Różnice w wartości delty dla różnych przedziałów wynikają z budowy liczb w standardzie IEEE 754. Odległości między kolejnymi różnymi liczbami są zależne od cechy liczby. (pogrubiony fragment) W danej cesze liczby występują tak samo gęsto. Zmiana cechy o 1 powoduje zmianę delty, a więc "zagęszczenia". W przypadku wzrostu cechy o 1, dwukrotnie wzrasta odległość między kolejnymi liczba – analogicznie w przypadku zmniejszenia cechy – odległość spada o połowę.

Zadanie 4

4.1 Znajdywanie $x + \frac{1}{x} \neq 1$

4.1.1 Treść

Znajdź eksperymentalnie w arytmetyce Float64 zgodnej ze standardem IEEE 754 (double) liczbę zmiennopozycyjną x w przedziale 1 < x < 2, taką, że $x + \frac{1}{x} \neq 1$

4.1.2 Rozwiązanie

W celu odnalezienia liczby spełniającej warunek zadania napisałem program, który w pętli dodawał do liczby 1 wartość *macheps* do momentu spełnienia warunku $x + \frac{1}{x} \neq 1$. Z racji tego, że zacząłem od 1.0, odnaleziona liczba jest też najmniejszą możliwą.

4.1.3 Wynik

Odnaleziona liczba to:

1.000000057228997

Bity:

0011111111110000000000000000000000011110101110010111111100101010

Zadanie 5

5.1 Iloczyn skalarny dwóch wektorów

5.1.1 Treść

Napisz program w języku Julia realizujący następujący eksperyment obliczania iloczynu skalarnego dwóch wektorów. Policz sumę na 4 sposoby.

5.1.2 Rozwiązanie

Zadanie polegało na obliczeniu iloczynu skalarnego dwóch zadanych wektorów przy pomocy 4 algorytmów. **Algorytm A** polegał na obliczeniu "w przód" sumując kolejne iloczyny składników**,** **Algorytm B** na obliczeniu "w tył" sumując iloczyny od końca, **Algorytm C** na obliczeniu dwóch sum, oddzielnie dla iloczynów dodatnich (posortowane od największych do najmniejszych), oddzielnie dla ujemnych (od największych do najmniejszych), a później sumowaniu sum częściowych, a **Algorytm D** na działaniu odwrotnym do Algorytmu C w przy liczeniu sum częściowych.

5.1.3 Wynik

Float32

Algorytm	**Wynik**
A	-4.999443e-01
B	-4.543457e-01
C	-5.000000e-01
D	-5.000000e-01

Float64

Algorytm	**Wynik**
A	1.025188e-10
B	-1.564331e-10
C	0.000000e+00
D	0.000000e+00

Rzeczywista wartość: −1.00657107000000e − 11

Jak widać, wyniki otrzymane w wyniku działania każdego z algorytmów różnią się znacząco od rzeczywistej wartości. Jeśli chodzi o analizę, to o ile wyniki *Float32* są bardzo rozbieżne, to wyniki dla arytmetyki podwójnej precyzji dały dobre przybliżenie.

Cennym wnioskiem z przeprowadzonego ćwiczenia jest fakt, że kolejność dodawania liczb ma wpływ na to, jaki wynik otrzymamy.

Zadanie 6

6.1 Oblicz wartość funkcji

6.1.1 Treść zadania

Policz w języku Julia w arytmetyce Float64 wartości następujących funkcji $f(x) = \sqrt{x^2 + 1} - 1$ oraz $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ dla kolejnych wartości argumentu x = 8^{-1}, 8^{-2}, 8^{-3}, ... Chociaż $f(x) = g(x)$ to komputer daje różne wyniki. Które z nich są wiarygodne, a które nie?

6.1.2 Rozwiązanie

W celu wyliczenia wartości funkcji stworzyłem w Julii funkcje, które w odpowiedniej kolejności wykonywały poszczególne operacje. Wartości zwracane przez poszczególne wywołania - z odpowiednim parametrem - zawarłem w poniższej tabeli.

6.1.3 Wynik

Parametr	Wartość $f(x)$	Wartość $g(x)$
8^{-1}	7.782219e-03	7.782219e-03
8^{-2}	1.220629e-04	1.220629e-04
8^{-3}	1.907347e-06	1.907347e-06
8^{-4}	2.980232e-08	2.980232e-08
8^{-5}	4.656613e-10	4.656613e-10
8^{-6}	7.275958e-12	7.275958e-12
8^{-7}	1.136868e-13	1.136868e-13
8^{-8}	1.776357e-15	1.776357e-15
8^{-9}	0.000000e+00	2.775558e-17
8^{-10}	0.000000e+00	4.336809e-19
8^{-11}	0.000000e+00	6.776264e-21
8^{-12}	0.000000e+00	1.058791e-22

Do pewnego momentu wyniki dla $f(x)$ i $g(x)$ są sobie równe, i – po porównaniu ich z wynikami wyliczeń w pakiecie matematycznym Wolfram Alpha - poprawne. Jednak, gdy, w przypadku $f(x)$, argument funkcji osiąga wartość 8^{-9}, funkcja zaczyna zwracać wartość 0. Wynika to prawdopodobnie z tego, że na koniec wykonujemy odejmowanie 1 od bardzo małego pierwiastka.

Zadanie 7

7.1 Wyznacz pochodną

7.1.1 Treść zadania

Skorzystać z wzoru do obliczenia w języku Julia w arytmetyce *Float64* przybliżonej wartości pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x_0 = 1$ oraz błędów.

7.1.2 Rozwiązanie

W celu wyliczenia wartości pochodnej stworzyłem w Julii funkcję wykorzystującą wzór

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

7.1.3 Wynik

Rzeczywista wartość tej pochodnej w punkcie 1 to około 0.116942281. Ta wartość posłużyła mi do obliczenia błędu. Poniżej zestawiłem wartości zwracane przez funkcję w Julii.

Dla dużych wartości przyrostu h

h	Wartość funkcji pochodnej	Błąd
2^{-1}	1.870441e+00	1.753499e+00
2^{-2}	1.107787e+00	9.908448e-01
2^{-3}	6.232413e-01	5.062990e-01
2^{-4}	3.704001e-01	2.534578e-01

Dla wartości bliskich realnej wartości

h	Wartość funkcji pochodnej	Błąd
2^{-26}	1.169423e-01	5.764546e-08
2^{-27}	1.169423e-01	3.529372e-08
2^{-28}	1.169423e-01	5.491394e-09

Dla bardzo małych wartości przyrostu h

h	Wartość funkcji pochodnej	Błąd
2^{-51}	0.000000e+00	1.169423e-01
2^{-52}	-5.000000e-01	6.169423e-01
2^{-53}	0.000000e+00	1.169423e-01
2^{-54}	0.000000e+00	1.169423e-01

Zmniejszając wartość przyrostu h do pewnego momentu osiągamy oczekiwane rezultaty – wynik bliższy prawdzie. W pewnym momencie natomiast przewagę nad dokładnością przejmuje błąd obliczeń arytmetyki. Prawdopodobnie wynika to z tego, że operujemy na bardzo małych liczbach coraz bliższych epsilonowi maszynowemu. Gdy wartości przyrostu osiągają epsilon maszynowe, wyniki przestają się zmieniać.

Chcąc uzyskać lepszy, dokładniejszy wynik, musielibyśmy zwiększyć dokładność arytmetyki.