Vadym Liss

257264

**Zadanie 1**

**Epsilon maszynowy**

*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.*

Wyniki programu w języku Julia wyznaczającego iteracyjnie epsilony maszynowe dla wszystkich dostępnych typów zmiennopozycyjnych:

|  |  |  |
| --- | --- | --- |
| Standard | Macheps wyliczony | Eps(Float) |
| Float16 | 0.000977 | 0.000977 |
| Float32 | 1.1920929e-7 | 1.1920929e-7 |
| Float64 | 2.220446049250313e-16 | 2.220446049250313e-16 |

Widzimy, że wartości otrzymane eksperymentalnie są podobne do tych, które są zwracane przez funkcje języka Julia.

Z kolei dane zawarte w pliku nagłówkowym float.h instalacji języka C:

|  |  |
| --- | --- |
| Funkcja | Zwracana wartość |
| FLT\_EPSILON | 1E-5 lub mniejsze |
| DBL\_EPSILON | 1E-9 lub mniejsze |
| LDBL\_EPSILON | 1E-9 lub mniejsze |

Precyzje arytmetyki:

|  |  |
| --- | --- |
| Standard | Zwracana wartość |
| Float16 | 0.00048828125 |
| Float32 | 5.960464477539063e-8 |
| Float64 | 1.1102230246251565e-16 |

**Wniosek:** Precyzja arytmetyki mówi o tym, ile cyfr znaczących (stanowiących mantysę w IEEE 754) jest reprezentowanych dokładnie, innymi słowy, na którym miejscu znajduje się ostatni bit reprezentowany dokładnie. Precyzja jest związana z długością mantysy (która wynosi kolejno 10, 23, 52 bity/ów dla Float16, Float32, Float64). Z pomocą powyższych obliczeń można zauważyć, że epsilon maszynowy różni się o jeden rząd wielkości w systemie dzisiętnym od precyzji poszczególnych typów, zatem mieści się w mantysie.

**Eta**

*Napisać program w języku Julia wyznaczający iteracyjnie liczbę maszynową 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))*

Wyniki programu w języku Julia wyznaczającego iteracyjnie liczbę maszynową eta dla wszystkich dostępnych typów zmiennopozycyjnych:

|  |  |  |
| --- | --- | --- |
| Standard | Eta wyliczona | Nextfloat(Float) |
| Float16 | 6.0e-8 | 6.0e-8 |
| Float32 | 1.0e-45 | 1.0e-45 |
| Float64 | 5.0e-324 | 5.0e-324 |

Widzimy, że wartości otrzymane eksperymentalnie są takie same jak i te, które są zwracane przez funkcje języka Julia. Zauważmy, że liczba eta jest równa liczbie .

|  |  |
| --- | --- |
| Funkcja | Zwracana wartość |
| floatmin(Float32) | 1.1754944e-38 |
| floatmin(Float64) | 2.2250738585072014e-308 |

**Wniosek:** Powyższe wartości są równe wartości dla odpowiedniej arytmetyki.

**Max**

*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: floatmax(Float16), floatmax(Float32), floatmax(Float64) oraz z danymi zawartymi w pliku nagłówkowym float.h dowolnej instalacji języka C.*

Wyniki programu w języku Julia wyznaczającego iteracyjnie liczbę MAX dla wszystkich dostępnych typów zmiennopozycyjnych:

|  |  |  |
| --- | --- | --- |
| Standard | Max wyliczony | Floatmax(Float) |
| Float16 | 6.55e4 | 6.55e4 |
| Float32 | 3.4028235e38 | 3.4028235e38 |
| Float64 | 1.7976931348623157e308 | 1.7976931348623157e308 |

Widzimy, że wartości otrzymane eksperymentalnie są podobne do tych, które są zwracane przez funkcje języka Julia.

|  |  |
| --- | --- |
| Format | MAX |
| Float32 | 3.4 \* |
| Float64 | 1.8 \* |

Patrząc na dane podane na wykładzie i porównując ich z wynikami naszego programu, jest banalnie oczywistym to, że maksymalna liczba wyliczona przez mnie pokrywa się z wartościami z tablicy z wykładu.

**Wniosek:** Arytmetyka IEEE-754 ma pewne ograniczenia, które trzeba brać pod uwagę, jeżeli chce się otrzymać dokładne wyniki obliczeń. Tzn. każdy typ zmiennopozycyjny arytmetyki IEEE 754 ma skończoną dokładność.

**Zadanie 2**

*Sprawdzić eksperymentalnie w języku Julia słuszność tego stwierdzenia dla wszystkich typów zmiennopozycyjnych Float16, Float32, Float64.*

Wyniki programu w języku Julia do sprawdzenia stwierdzenia Kahan’a dla wszystkich dostępnych typów zmiennopozycyjnych:

|  |  |  |
| --- | --- | --- |
| Standard | Wyrażenie wyliczone | Eps(Float) |
| Float16 | -0.000977 | 0.000977 |
| Float32 | 1.1920929e-7 | 1.1920929e-7 |
| Float64 | -2.220446049250313e-16 | 2.220446049250313e-16 |

**Wniosek**: Jeżeli na wyrażenie nałożyłoby się wartość bezwzględną, to stwierdzenie jest jak najbardziej prawdziwe.

**Zadanie 3**

*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 δ = .*

Z kolei przedstawiam wyniki do wybranej liczby kroków i w zależności od początku:

|  |  |  |
| --- | --- | --- |
| Początek | krok | Check\_delta |
| 1 |  | 0011111111110000000000000000000000000000000000000000000000000001  0011111111110000000000000000000000000000000000000000000000000010  0011111111110000000000000000000000000000000000000000000000000011  0011111111110000000000000000000000000000000000000000000000000100  0011111111110000000000000000000000000000000000000000000000000101  0011111111110000000000000000000000000000000000000000000000000110  0011111111110000000000000000000000000000000000000000000000000111  0011111111110000000000000000000000000000000000000000000000001000  0011111111110000000000000000000000000000000000000000000000001001 |
| 0.5 |  | 0011111111100000000000000000000000000000000000000000000000000001  0011111111100000000000000000000000000000000000000000000000000010  0011111111100000000000000000000000000000000000000000000000000011  0011111111100000000000000000000000000000000000000000000000000100  0011111111100000000000000000000000000000000000000000000000000101  0011111111100000000000000000000000000000000000000000000000000110  0011111111100000000000000000000000000000000000000000000000000111  0011111111100000000000000000000000000000000000000000000000001000  0011111111100000000000000000000000000000000000000000000000001001 |
| 2 |  | 0100000000000000000000000000000000000000000000000000000000000001  0100000000000000000000000000000000000000000000000000000000000010  0100000000000000000000000000000000000000000000000000000000000011  0100000000000000000000000000000000000000000000000000000000000100  0100000000000000000000000000000000000000000000000000000000000101  0100000000000000000000000000000000000000000000000000000000000110  0100000000000000000000000000000000000000000000000000000000000111  0100000000000000000000000000000000000000000000000000000000001000  0100000000000000000000000000000000000000000000000000000000001001 |
| 3 |  | 0100000000001000000000000000000000000000000000000000000000000001  0100000000001000000000000000000000000000000000000000000000000010  0100000000001000000000000000000000000000000000000000000000000011  0100000000001000000000000000000000000000000000000000000000000100  0100000000001000000000000000000000000000000000000000000000000101  0100000000001000000000000000000000000000000000000000000000000110  0100000000001000000000000000000000000000000000000000000000000111  0100000000001000000000000000000000000000000000000000000000001000  0100000000001000000000000000000000000000000000000000000000001001 |

**Wniosek:** Na podstawie powyższych eksperymentów można stwierdzić, że liczby w podanych przedziałach są równomiernie rozłożone. Krok dla danego przedziału zależy od cechy - rośnie wraz z jej wzrostem. Stały odstęp między liczbami z podanych przedziałów, wynika z niezmiennej liczby bitów mantysy. Można zatem stwierdzić, że w każdym takim przedziale można reprezentować tyle samo liczb. Dla przedziału [1,2], równomierne rozmieszczenie dla kroku *δ =* , dla [1/2, 1], równomierne rozmieszczenie dla kroku *δ =* , dla przedziału [2,4], równomierne rozmieszczenie dla kroku *δ =* .

**Zadanie 4**

1. *Znajdź eksperymentalnie w arytmetyce Float64 zgodnej ze standardem IEEE 754 (double) liczbę zmiennopozycyjną x w przedziale 1 < x < 2, taką, że x ∗ (1/x) ≠ 1; tj. fl(xfl(1/x)) ≠ 1*

Wynik zwrócony przez nasz program: 1.5000000000000002.

*b)* Znajdź najmniejszą taką liczbę

Eksperymentalnie ustaliłem, że taka liczba wynosi: 1.000000057228997. **Wniosek:** Dowiadujemy się o ograniczeniach arytmetyki IEEE-754, ważniejszym z których jest, że

liczby przedstawiane dokładnie w systemie dziesiętnym ma nieskończone rozwinięcie w systemie binarnym, przez co one nie mogą być dokładnie reprezentowane. Przez to pojawiają się błędy, takie jak w tym

zadaniu.

**Zadanie 5**

*Napisać program w języku Julia realizujący następujący eksperyment obliczania iloczynu skalarnego dwóch wektorów:*

*x = [2.718281828, −3.141592654, 1.414213562, 0.5772156649, 0.3010299957]*

*y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049].*

1. **“w przód”**

|  |  |  |
| --- | --- | --- |
| Standard | s | Różnica |
| Float32 | -0.12593332 | 0.12593331932014976 |
| Float64 | 1.0251881368296672e-10 | 1.1258452438296672e-10 |

1. **“w tył”**

|  |  |  |
| --- | --- | --- |
| Standard | s | Różnica |
| Float32 | -0.23105995 | 0.2310599535603002 |
| Float64 | -1.5643308870494366e-10 | 1.4636737800494365e-10 |

1. **od największego do najmniejszego**

|  |  |  |
| --- | --- | --- |
| Standard | s | Różnica |
| Float32 | -0.25 | 0.2499999999899343 |
| Float64 | 0.0 | 1.0065710700000004e-11 |

1. **od najmniejszego do największego**

|  |  |  |
| --- | --- | --- |
| Standard | s | Różnica |
| Float32 | -0.25 | 0.2499999999899343 |
| Float64 | 0.0 | 1.0065710700000004e-11 |

**Wniosek:** Poznajemy ważną własność arytmetyki IEEE-754, a konkretniej to, że dodawanie w niej nie jest przemienne. Także zauważamy, że wykorzystane algorytmy mają wpływ na końcowe wyniki.

**Zadanie 6**

Policzyć w języku Julia w arytmetyce Float64 wartości następujących funkcji

f(x) = − 1

g(x) = /( + 1)

dla kolejnych wartości argumentu x = , , , . . ..

W podanej tablicy mamy wyniki F(x) oraz odpowiednio G(x):

|  |  |  |
| --- | --- | --- |
| x | F(x) | G(x) |
| 0.125 | 0.0077822185373186414 | 0.0077822185373187065 |
| 0.015625 | 0.00012206286282867573 | 0.00012206286282875901 |
| 0.001953125 | 1.9073468138230965e-6 | 1.907346813826566e-6 |
| 0.000244140625 | 2.9802321943606103e-8 | 2.9802321943606116e-8 |
| 3.0517578125e-5 | 4.656612873077393e-10 | 4.6566128719931904e-10 |
| 3.814697265625e-6 | 7.275957614183426e-12 | 7.275957614156956e-12 |
| 5.960464477539063e-8 | 1.7763568394002505e-15 | 1.7763568394002489e-15 |
| 4.76837158203125e-7 | 1.1368683772161603e-13 | 1.1368683772160957e-13 |
| 7.450580596923828e-9 | 0.0 | 2.7755575615628914e-17 |
| 9.313225746154785e-10 | 0.0 | 4.336808689942018e-19 |
| 1.1641532182693481e-10 | 0.0 | 6.776263578034403e-21 |
| 1.4551915228366852e-11 | 0.0 | 1.0587911840678754e-22 |

**Wniosek:** Od pewnego momentu (już dla n = 9) funkcja f zaczyna zwracać 0.0. Powinno się unikać odejmowania czy dodawania liczb bardzo bliskich sobie, ponieważ ryzykujemy utratą cyfr znaczących, a przez to niedokładnym wynikiem.

**Zadanie 7**

*Przybliżoną wartość pochodnej f(x) w punkcie x można obliczyć za pomocą następującego wzoru:*

*Skorzystać ze wzoru do obliczenia w języku Julia w arytmetyce Float64 przybliżonej wartości pochodnej funkcji f(x) = sin x + cos 3x w punkcie= 1 oraz błędów | − | dla h = (n = 0, 1, 2, . . ., 54).*

W podanej tablicy podaję to, jak zależy błąd od wyliczonych wartości:

|  |  |
| --- | --- |
| Wartość wyliczona | Błąd |
| 2.0179892252685967 | 1.9010469435800585 |
| 1.8704413979316472 | 1.753499116243109 |
| 1.1077870952342974 | 0.9908448135457593 |
| 0.6232412792975817 | 0.5062989976090435 |
| 0.3704000662035192 | 0.253457784514981 |
| 0.24344307439754687 | 0.1265007927090087 |
| 0.18009756330732785 | 0.0631552816187897 |
| 0.1484913953710958 | 0.03154911368255764 |
| 0.1248236929407085 | 0.007881411252170345 |
| 0.11891225046883847 | 0.001969968780300313 |
| 0.11743474961076572 | 0.0004924679222275685 |
| 0.11718851362093119 | 0.0002462319323930373 |
| 0.11706539714577957 | 0.00012311545724141837 |
| 0.11700383928837255 | 6.155759983439424e-5 |
| 0.11697306045971345 | 3.077877117529937e-5 |
| 0.11695767106721178 | 1.5389378673624776e-5 |
| 0.11694997636368498 | 7.694675146829866e-6 |
| 0.11694612901192158 | 3.8473233834324105e-6 |
| 0.1169442052487284 | 1.9235601902423127e-6 |
| 0.11694324295967817 | 9.612711400208696e-7 |
| 0.11694276239722967 | 4.807086915192826e-7 |
| 0.11694252118468285 | 2.394961446938737e-7 |
| 0.116942398250103 | 1.1656156484463054e-7 |
| 0.11694233864545822 | 5.6956920069239914e-8 |
| 0.11694231629371643 | 3.460517827846843e-8 |
| 0.11694228649139404 | 4.802855890773117e-9 |
| 0.11694222688674927 | 5.480178888461751e-8 |
| 0.11694216728210449 | 1.1440643366000813e-7 |
| 0.11694192886352539 | 3.5282501276157063e-7 |
| 0.11694145202636719 | 8.296621709646956e-7 |
| 0.11693954467773438 | 2.7370108037771956e-6 |
| 0.1169281005859375 | 1.4181102600652196e-5 |
| 0.116943359375 | 1.0776864618478044e-6 |
| 0.11688232421875 | 5.9957469788152196e-5 |
| 0.1168212890625 | 0.0001209926260381522 |
| 0.116943359375 | 1.0776864618478044e-6 |
| 0.11669921875 | 0.0002430629385381522 |
| 0.1162109375 | 0.0007313441885381522 |
| 0.1171875 | 0.0002452183114618478 |
| 0.11328125 | 0.003661031688538152 |
| 0.109375 | 0.007567281688538152 |
| 0.09375 | 0.023192281688538152 |
| 0.125 | 0.008057718311461848 |
| 0.0 | 0.11694228168853815 |
| -0.5 | 0.6169422816885382 |

Czasami nie obserwujemy poprawy przybliżenia, ponieważ wartość wyrażenia 1+h zbliża się do 1, aż do momentu, gdy h jest na tyle małe, że otrzymujemy coś takiego jak 1+h = 1, wynikającego z ograniczenia precyzji.

**Wniosek**: Najlepiej unikać dodawania kolejnych cyfr, różniących się rzędem, ponieważ to działanie niesie ze sobą ryzyko utraty dokładności wyniku.