Kajetan Bilski 244942

Obliczenia naukowe

Lista 1

Sprawozdanie

Zad. 1.

W zadaniu należy napisać kilka funkcji w julii, które iteracyjnie znajdą szukane liczby. Inne szukane liczby trzeba znaleźć wywołując proste funkcje w julii i c. Na końcu trzeba wyniki odpowiednio porównać i znaleźć zależności.

Napisałem 2 programy obliczające różne wartości floatów w julii i c (zad1.jl i eps.c). W tabeli umieszczam wyniki.

|  |  |
| --- | --- |
| Macheps dla Float16 | 0.000977 |
| Macheps dla Float32 | 1.1920929e-7 |
| Macheps dla Float64 | 2.220446049250313e-16 |
| Eta dla Float16 | 6.0e-8 |
| Nextfloat(Float16(0.0)) | 6.0e-8 |
| Eta dla Float32 | 1.0e-45 |
| Nextfloat(Float32(0.0)) | 1.0e-45 |
| Eta dla Float64 | 5.0e-324 |
| Nextfloat(Float64(0.0)) | 5.0e-324 |
| Floatmin(Float32) | 1.1754944e-38 |
| Floatmin(Float64) | 2.2250738585072014e-308 |
| Max dla Float16 | 6.55e4 |
| Floatmax(Float16) | 6.55e4 |
| Max dla Float32 | 3.4028235e38 |
| Floatmax(Float32) | 3.4028235e38 |
| Max dla Float64 | 1.7976931348623157e308 |
| Floatmax(Float64) | 1.7976931348623157e308 |
| FLT\_EPSILON | 1.192093e-007 |
| DBL\_EPSILON | 2.220446e-016 |

Macheps’y uzyskałem dzieląc 1.0 na 2 tak długo jak 1 + x > 1.0. Podobnie dla ety, tylko x > 0.0. Max’y uzyskałem zaczynając z x = 1.0 i mnożąc x^2 tak długo jak nie staje się to nieskończonością, następnie zacząłem dodawać w pętli do x x\*2^-i, gdzie i = 1,2,3,… aż do momentu kiedy x + x\*2^-i == x. W ten sposób zarówno cecha jak i mantysa zostały zmaksymalizowane.

Jak widać epsilon maszynowy jest mniejszy dla typów z większą ilością bitów. Wyliczone iteracyjnie epsilony 32- i 64-bitowe zgadzają się ze stałymi z float.h.

Można zauważyć że macheps = 2^-t gdzie t to ilość bitów mantysy. Na wykładzie zdefiniowany epsilon = 0.5\*2^1-t = 2^-t, czyli macheps = epsilon.

|  |  |  |
| --- | --- | --- |
|  | Eta | MINsub |
| Float32 | 1.0e-45 | 1.4·10−45 |
| Float64 | 5.0e-324 | 4.9·10−324 |

Eta jest blisko MINsub ale nie jest taka sama.

|  |  |  |
| --- | --- | --- |
|  | Floatmin | MINnor |
| Float32 | 1.1754944e-38 | 1.2·10−38 |
| Float64 | 2.2250738585072014e-308 | 2.2·10−308 |

Floatmin = MINnor

Zad. 2.

W zadaniu trzeba w julii obliczyć 3(4/3-1)-1 na floatach 16-, 32- i 64-bitowych i porównać z epsilonami maszynowymi z zadania 1.

Uzyskane wartości (zad2.jl):

|  |  |  |
| --- | --- | --- |
|  | 3(4/3-1)-1 | Macheps |
| Float16 | -0.000977 | 0.000977 |
| Float32 | 1.1920929e-7 | 1.1920929e-7 |
| Float64 | -2.220446049250313e-16 | 2.220446049250313e-16 |

Jak widać wartości bezwzględne się zgadzają ale w float16 i float64 bit znaku się przekręcił.

Podany wzór więc nie zawsze daje macheps, ale pomaga w jego znalezieniu.

Zad. 3.

W tym zadaniu trzeba użyć funkcji bitstring() języka julia, żeby przeanalizować bity kolejnych floatów i pokazać prawdziwość założenia z treści zadania.

julia> x = 1e0

1.0

julia> bitstring(x)

"0011111111110000000000000000000000000000000000000000000000000000"

julia> x += 2^-52

1.0000000000000002

julia> bitstring(x)

"0011111111110000000000000000000000000000000000000000000000000001"

julia> x += 2^-52

1.0000000000000004

julia> bitstring(x)

"0011111111110000000000000000000000000000000000000000000000000010"

julia> bitstring(2e0 - 2^-52)

"0011111111111111111111111111111111111111111111111111111111111111"

Dodanie do liczby z przedziału [1,2-2^-52) 2^-52 dodaje 1 do mantysy (tak jakby była unsigned int).

Dla [1/2,1] δ = 2^-53 a dla [2,4] δ = 2^-51, ponieważ x = m\*2^c, gdzie m to liczba reprezentowana przez mantysę, a c to cecha. Kiedy cecha zwiększa się o 1 to waga każdego bitu mantysy podwaja się. Dla każdego przedziału [2^n, 2^(n+1)] odległość między kolejnymi liczbami δ = 2^(n-52).

Zad. 4.

W tym zadaniu trzeba w języku julia znaleźć liczbę x we Float64 taką, że 1 < x < 2 i x\*(1/x) != 1.0. Należy znaleźć też najmniejszą taką liczbę.

Do tego celu użyłem w pętli funkcji nextfloat().

Najmniejsza taka liczba to 1.000000057228997.

Pokazuje to że niedokładności w arytmetyce Float64 szybko się nakładają powodując błędy w obliczeniach, co jest najbardziej dotkliwe przy sprawdzaniu równości. Najmniejsza znaleziona liczba gdzie pojawia się wyżej opisany błąd jest niewiele większa od 1, co może być dowodem na dużą częstotliwość występowania takich liczb.

Zad. 5.

|  |  |  |
| --- | --- | --- |
|  | Float32 | Float64 |
| W przód | -0.4999443 | 1.0251881368296672e-10 |
| W tył | -0.4543457 | -1.5643308870494366e-10 |
| Od największego do najmniejszego | -0.5 | 0.0 |
| Od najmniejszego do największego | -0.5 | 0.0 |

Widać że dokładność float’a 32 zostawia wiele do życzenia. W przypadku double’a dokładność jest lepsza, szczególnie w przypadku liczenia „w tył”.

Zad. 6.

W tym zadaniu trzeba w policzyć w julii funkcje f(x) i g(x) na float64 dla podanych argumentów i porównać wyniki.

|  |  |  |
| --- | --- | --- |
| n | f(8^-n) | g(8^-n) |
| 1 | 0.0077822185373186414 | 0.0077822185373187065 |
| 2 | 0.00012206286282867573 | 0.00012206286282875901 |
| 3 | 1.9073468138230965e-6 | 1.907346813826566e-6 |
| 4 | 2.9802321943606103e-8 | 2.9802321943606116e-8 |
| 5 | 4.656612873077393e-10 | 4.6566128719931904e-10 |
| 6 | 7.275957614183426e-12 | 7.275957614156956e-12 |
| 7 | 1.1368683772161603e-13 | 1.1368683772160957e-13 |
| 8 | 1.7763568394002505e-15 | 1.7763568394002489e-15 |
| 9 | 0.0 | 2.7755575615628914e-17 |
| 10 | 0.0 | 4.336808689942018e-19 |
| 11 | 0.0 | 6.776263578034403e-21 |
| 12 | 0.0 | 1.0587911840678754e-22 |
| 13 | 0.0 | 1.6543612251060553e-24 |
| 14 | 0.0 | 2.5849394142282115e-26 |
| 15 | 0.0 | 4.0389678347315804e-28 |

Wynik zwracany przez wolfram alpha dla f(8^-15) = 1.4210854715201902743226617063486336926212230994517922×10^(-14)

Jak widać wartości funkcji f(x) i g(x) wyliczane w julii są bardzo blisko siebie aż do x = 8^-8. Od x = 8^-9 obliczanie funkcji w sposób f daje 0, podczas gdy g wciąż daje dodatnie rezultaty chociaż już nie takie bliskie rzeczywistości.

Zad. 7.

W tym zadaniu trzeba zaimplementować w julii funkcję, która oblicza różnicę między faktyczną pochodną funkcji f, a funkcją, która ją przybliża, gdzie f(x) = sin x + cos 3\*x, f’(x) = cos x – 3 \* sin 3\*x. Pochodną przybliżamy wzorem (f(x+h) – f(x))/h dla h = 2^-n, n = 0..54. Obliczamy to tylko dla x = 1.



Błąd zmniejsza się razem z n aż do n = 28, potem błąd znowu się zwiększa aż do n = 52. Od n = 53 h < macheps i x + h = x. Najwyraźniej dla mniejszych n błąd wynika z różnicy pomiędzy x i x + h, a dla większych n z niedokładności Float64 podobnych do tych z 4. zadania.