Obliczenia naukowe
Sprawozdanie z listy 1

Wrocław, 14 października 2017

1.1 Opis problemu

W zadaniu należy napisać trzy programy. Pierwszy do obliczania epsilonu maszynowego, drugi do obliczania liczby eta i trzeci do obliczania liczby max. Programy mają być zrealizowane dla dostępnych typów zmiennopozycyjnych tj. Float16, Float32, Float64.

1.2 Rozwiązanie

Aby wyznaczyć epsilon maszynowy została utworzona funkcja, w której jest zasadnicza pętla: dziel epsilon/2 dopóki x+epsilon/2 > x (dla danych wejściowych: x=Float16/32/64(1) i epsilon=Float16/32/64(1)).

Druga funkcja do obliczania liczby eta polegała na sprawdzeniu w pętli czy eta/2 > 0 jeśli tak wykonywane było działanie eta/2. W przeciwnym razie pętla się kończyła i był zwracany wynik.

Trzecia funkcja do obliczania największej liczby dodatniej. Pętla mnoży *2 w danej precyzji, jeśli wartość nie przekroczy zakresu. Do sprawdzania jest użyta funkcja isinf().

Każda z trzech funkcji została przetestowana dla trzech następujących typów: Float16, Float32, Float64.

1.3 Wyniki programu i wnioski

Podpunkt pierwszy:

Typ	Epsilon maszynowy	eps()
Float16	0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	2.220446049250313e-16	2.220446049250313e-16

Podpunkt drugi:

Typ	eta	naxtfloat()
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Podpunkt trzeci:

Typ	max	realmax()
Float16	6.55e4	6.55e4
Float32	3.4028235e38	3.4028235e38
Float64	1.7976931348623157e308	1.7976931348623157e308

Wyniki przedstawione w tabelach potwierdzają poprawność działania algorytmów w programie.

2.1 Opis problemu

W tym zadaniu należy udowodnić (przedstawiając adekwatne wyniki programu) poprawność stwierdzenia Kahana mówiącego, że epsilon maszynowy w arytmetyce zmiennopozycyjnej można policzyć za pomocą wyrażenia 3(3/4 – 1) – 1.

2.2 Rozwiązanie

Funkcja programu polega na obliczeniu równania x = 3(4/3)-1 dla zadanych trzech typów zmiennopozycyjnych: Float16, Float32, Float64 i ich wyświetlenie.

2.3 Wyniki programu i wnioski

Typ	Wynik programu
Float16	-0.0
Float32	-2.220446e-16
Float64	-2.220446049250313e-16

Wyniki przedstawione w tabeli pokazują poprawność wyłącznie dla Float64. W pozostałych przypadkach arytmetyka nie była wystarczająca dla tego obliczenia.

3.1 Opis problemu

W tym zadaniu należy sprawdzić w arytmetyce float64 czy liczby zmiennopozycyjne są równomiernie rozmieszczone w [1;2] oraz jak są one rozmieszczone w przedziale [0,5;5] i w przedziale [2;4].

3.2 Rozwiązanie

W programie została zastosowana funkcja , która dla poszczególnych liczb z danych przedziałów wyświetla ich reprezentacje bitowe. Aby uzyskać kolejną liczbę użyto x = nextfloat(x).

3.3 Wyniki programu i wnioski

Przedział	Reprezentacja bitowa
[1;2]	0011111111110000000000000000000000000000000000000000000000000000
[1;2]	0011111111110000000000000000000000000000000000000000000000000001
[1;2]	0011111111110000000000000000000000000000000000000000000000000010
[1;2]	0011111111110000000000000000000000000000000000000000000000000011
[0,5;5]	0011111111100000000000000000000000000000000000000000000000000000
[0,5;5]	0011111111110000000000000000000000000000000000000000000000000000
[0,5;5]	0011111111110000000000000000000000000000000000000000000000000001
[0,5;5]	0011111111110000000000000000000000000000000000000000000000000010
[2;4]	0100000000000000000000000000000000000000000000000000000000000000
[2;4]	0100001100100000000000000000000000000000000000000000000000000010
[2;4]	0100001100110000000000000000000000000000000000000000000000000001
[2;4]	0100001100111000000000000000000000000000000000000000000000000001

Dzięki przedstawionemu wyżej zestawieniu możemy zobaczyć jak są rozmieszczone liczby w danych przedziałach.

4.1 Opis problemu

W tym zadaniu należy znaleźć w arytmetyce float64 liczbę w przedziale od 1 do 2 taką, że $x * (1/x) \neq 1$ i minimalną taką liczbę.

4.2 Rozwiązanie

W tym programie jest zastosowana funkcja, która sprawdza w pętli czy liczba jest mniejsza od 2 i czy jeszcze jest nie spełniony warunek zadania (tz. sprawdza x * (1/x) = 1), jeżeli będzie różne zwróci szukaną liczbę jeśli nie x = nestfloat(x). Dane wejściowe to x=Float64(1)

4.3 Wyniki programu i wnioski

Wynikiem jest liczba spełniająca warunki zadania: 1.000000057228997, tz. jest najmniejszą liczbą arytmetyki float64 z przedziału [1;2].

5.1 Opis problemu

W zadaniu należy obliczyć iloczyn skalarny dwóch wektorów na cztery różne sposoby.

5.2 Rozwiązanie

W zadaniu zostały zrealizowane algorytmy w języku Julia, na podstawie tych podanych w polecaniu. Realizowana jest pętla wykonująca iloczyny.

5.3 Wyniki programu i wnioski

Podpunkt	Float32	Float64
1)	-0.4999443	1.0251881368296672e-10
2)	-0.4543457	-1.5643308870494366e-10
3)	-0.5	0.0

Otrzymaliśmy różne wyniki spośród, których żaden nie jest poprawny.

6.1 Opis problemu

W tym zadaniu należy w arytmetyce float64 wartości funkcji $f(x) = \sqrt{(x*+1)}–1$ oraz funkcji $g(x) = x*/(\sqrt{(x*+1)}+1)$.

6.2 Rozwiązanie

Obliczenia zostały wykonane następująco w języku Julia:
f(x)
(sqrt(x^2+1)-1)
i
g(x)
x^2/(sqrt(x^2+1)+1)

6.3 Wyniki programu i wnioski

	f(x)	g(x)
$x=8^{-1}$	0.0077822185373186414	0.0077822185373187065
$x=8^{-2}$	0.00012206286282867573	0.00012206286282875901
$x=8^{-3}$	1.9073468138230965e-6	1.907346813826566e-6
$x=8^{-4}$	2.9802321943606103e-8	2.9802321943606116e-8
$x=8^{-5}$	4.656612873077393e-10	4.6566128719931904e-10
$x=8^{-6}$	7.275957614183426e-12	7.275957614156956e-12
$x=8^{-7}$	1.1368683772161603e-13	1.1368683772160957e-13
$x=8^{-8}$	1.7763568394002505e-15	1.7763568394002489e-15
$x=8^{-9}$	0.0	2.7755575615628914e-17
$x=8^{-10}$	0.0	4.336808689942018e-19

Wyniki jednoznacznie pokazują, że to funkcja g(x) zapewnia lepszą precyzję obliczeń.

7.1 Opis problemu

W tym zadaniu należy obliczyć w arytmetyce float64 przybliżoną wartość pochodnej funkcji f(x) = sinx + cos3x w punkcie xo=1, oraz błędów pomiędzy właściwą wartością tej pochodnej, a przybliżoną wartością.

7.2 Rozwiązanie

Do rozwiązania tego zadanie został użyty program z7l1.jl, który oblicza przybliżoną wartość pochodnej oraz błąd pomiędzy właściwą a przybliżoną wartością tej pochodnej.

7.3 Wyniki programu i wnioski

h	Błąd
2^{-1}	1.9010469435800585
2^{-2}	1.753499116243109
2^{-3}	0.9908448135457593
2^{-13}	0.0004924679222275685
2^{-14}	0.0002462319323930373
2^{-15}	0.00012311545724141837
2^{-16}	6.155759983439424e-5
2^{-52}	0.6169422816885382
2^{-53}	0.11694228168853815
2^{-54}	0.11694228168853815

h+1	Błąd
2^{-1}	0.42768665270466855
2^{-2}	0.6121437007991494
2^{-3}	1.3387385610070992
2^{-13}	1.9008520854668745
2^{-14}	1.9009495263667628
2^{-15}	1.9009982379344046
2^{-16}	1.9010225914975019
2^{-52}	1.901046943580058
2^{-53}	1.9010469435800585
2^{-54}	1.9010469435800585

Na podstawie wyników widać, że używając wzoru na przybliżoną wartość pochodnej najdokładniejsze wyniki można uzyskać na końcu przedziału . Można zauważyć że wartości h+1 dla n należącego do przedziału [1;54] są niemalejące.