17.10.2019

Obliczenia Naukowe

Lista Nr 1(laboratorium)

Piotr Popis

- *Zadanie 1*

1.1 Wyznaczanie macheps(epsilon maszynowy)

1.1.1 Opis problemu

Epsilon maszynowy(macheps) to najmniejsza, dodatnia liczba taka, że fl(1.0+ macheps) > 1.0. Wyznaczenie macheps dla wszystkich wyznaczonych typów zmiennopozycyjnych zgodnych z standardem IEEE 754: Float16(half), Float32(single) i Float64(dobule) wymaga od nas wymyślenia odpowiedniego podejścia do wyznaczenia minimalnej, zauważalnej dla komputera liczby.

1.1.2 Rozwiązanie problemu

Podejście iteracyjnie jest odpowiednią drogą do wyznaczenia takiej liczby. Początkowo nadajemy macheps' owi wartość 1.0. Następnie macheps(początkowo 1.0) dzielimy iteracyjnie w pętli przez 2 dopóki warunek (1.0+macheps*0.5)>1.0 jest spełniony. Warunek wynika z tego, iż bada on jak wygląda macheps w kolejnej iteracji. Jeśli KOLEJNY machpes nie spełnia wyżej wymienionego warunku znaczy to, że poprzedni(aktualny) epsilon maszynowy jest poprawny i NAJMNIEJSZY, wtedy program przerywa iterowanie i zwraca szukany macheps. Rozwiązanie w języku Julia w załączonym pliku ZIP w module exercise1.

1.1.3 Wyniki

Typ zmiennopozycyjny	macheps	eps()	Float.h value
Float16	0.0009765625	0.000977	---
Float32	1.1920928955078125e-7	1.1920929e-7	1.1920928955078125e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.2204460492503131e-16

1.1.4 Wnioski

1.1.4.1 *Związek liczby macheps z precyzją arytmetyki*

Jeśli liczba nie może zostać zapisana w rozwinięciu dwójkowym na x bitach to musi ona zostać zastąpiona przez liczbę posiadającą taką reprezentajcę. Tzw błąd przybliżenia. Macheps to liczba, która stanowi różnicę pomiędzy 1, a kolejną wyznaczalną większą liczbą. Zatem możemy ją potraktować jako swoistą miarę precyzji. (Gęstość liczb ?). Macheps jest odwrotnie proporcjonalny do precyzji obliczeń – tym większy macheps tym mniejsza precyzja działań.

1.1.4.3 Ogólne

Dzielenie przez 2 to bitowe przesunięcie w prawo. Czyli przesuwamy do momentu rozpoznania liczby jako zero maszynowe.   Wartości nieskończeniewielu liczb są po prostu zaookrąglane. Macheps wyznaczony iteracyjnie jest bardziej precyzyjny niż wartość zwracana przez funkcję eps() i tak samo precyzyjny jak wartości z biblioteki float.h. Macheps, można nazwać miarą precyzji.

1.2 Wyznaczenie eta

1.2.1 Opis problemu

Kolejnym problemem jest wyznaczenie liczby eta. Eta to minimalna liczba spełniająca warunek eta > 0.0, dla wszystkich typów zmiennopozycyjnych: Float16, Float 32, Float 64 zgodnych ze standardem IEEE 754(half, single, double).Wyznaczone wartosci porównać z wartośćią metody nextFloat() dla każdego z typów.

1.2.2 Rozwiązanie problemu

Początkowo narzucamy wartośc ety na 1.0. I w kolejnych iteracjach pętli zmniejszamy jej wartość dwukrotnie w pętli z warunkiem eta* 0.5 > 0.0 . Przez co tak jak w poprzednim zadaniuwyznaczamy taką wartość, która po kolejnym przesunięciu bitowym daje zero maszynowe. Poprzednia wartosc od zera maszynowego to nasza minimalna ETA.

1.2.3 Wyniki

FTP	Eta	Nextfloat(0.0)
Float16	5.960464477539063e-8	6.0e-8
Float32	1.401298464324817e-45	1.0e-45
Float64	5.0e-324	5.0e-324

1.2.4 Wnioski

2.4.1 Jaki związek ma liczba eta z liczbą MIN_{sub}?

Eta to najmniejsza liczba zmiennoprzecinkowa różna od zera, którą możemy zapisać w systemie zmiennopozycyjnym. Liczba eta jest zdenormalizowana. Oznacza to, że wszystkie bity cechy mają wartość 0. W liczbie eta ostatni bit mantysy wynosi 1. Eta to MIN_{sub}.

2.4.2 Co zwracają funkcje floatmin(Float32) i floatmin(Float64) i jaki jest związek zwracanych wartości z liczbą MIN_{nor} (zob. wykład lub raport [1])?

Funkcja floatmin zwraca liczbę, która stanowi MIN_{nor} jest to znormalizowane MIN_{sub} . Znaczy to, że w cesze pojawia się 1.

2.4.3 Ogólne

*Wartość E*ta wyznaczona iterycyjnie jest bardziej precyzyjna niż wartość funkcji nextfloat(). Wartości zwaracane przez nextfloat() to zaookrąglenia.

1.3 Wyznaczanie MAX

1.3.1 Opis problemu

Wartość Max to największa liczba przed *Inf* możliwa do zapisania dla wszystkich typów zmiennopozycyjnych(Float16, Float 32, Float64) Zgodnych ze standardem IEEE 745.

1.3.2 Rozwiązanie problemu

Wartość Max wyliczamy przy użyciu funkcji isinf().
Nadajemy Max wartość 1.0 i dopóty 2 * Max jest rożny od Inf przypisujemy Max jej dwukrotność 2Max. Gdy 2*max wynosi Inf przerywamy iterowanie i zwracamy szukamy max.

1.3.3 Wyniki

FTP	Max	floatmax()	Float.h value
Float16	6.55e4	6.55e4	--
Float32	3.4028235e38	3.4028235e38	3.4028234663852886e+38
Float64	1.7976931348623157e308	1.7976931348623157e308	1.7976931348623157e+308

1.3.4 Wnioski

Wartości uzyskane w iteracyjnym podejściu są takie same jak wyniki rzeczywiste co znaczy że zostały wyznaczone poprawnie.

- *Zadanie 2*

2.1 Opis problemu

Dowód słuszności Twierdzenia Kahana, tj
Epsilon maszynowy możemy otrzymać obliczając wyrażenie 3.0*(4.0/3.0-1.0)-1.0 dla konkretnej arytmetyki.

2.2 Rozwiązanie problemu

Wykonujemy podane działanie w arytmetykach Float16, Float32, Float64 zgodnych ze standatdem IEEE 754 w języku Julia.

2.3 Wyniki

FTP	Kahan’s macheps	eps()
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

2.4 Wnioski

Wzór Kahan’a okazuje się być poprawny dla podanych typów zmiennopozycyjnych co do wartości bezwzględnej. Jednakże znaki dla Float16, Float 64 nie zgadzają się. Przy używaniu wzoru Kahan’a trzeba pamiętać o możliwości otrzymania błędnego znaku przed liczbą.

- *Zadanie 3*

3.1 Opis problemu

Eksperymentalne sprawdzenie w arytmetyce Float64, iż liczby zmiennopozycyjne są równomiernie rozmieszczone w przedziale [1,2] z krokiem $\delta = 2^{-52}$. Innymi słowy, każda liczba zmiennopozycyjna pomiędzy 1 i 2 może być przestawiona jako:

$$x = 1 + k * \delta$$

gdzie k = 1, … , 2^52 -1

Sprawdzić również w przedziałach [0.5,1] orax [2,4].

3.2 Rozwiązanie problemu

Aby sprawdzić rozmieszecznie wykorzystujemy fukncję bitstring(). Dla podanego zakresu k od 1 do (((δ)^(-1))-1) (dla uproszczenia 5) drukujemy kolejne wartosci (Float64(1.0 + k*δ))) liczby. Równomierne przesunięcia bitowe mogą świadczyć równomierności rozmieszczenia.

3.3 Wyniki

[1,2] $\delta = 2^{-52}$

0011111111110000000000000000000000000000000000000000000000000001
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000011
0011111111110000000000000000000000000000000000000000000000000100
0011111111110000000000000000000000000000000000000000000000000101

[0.5, 1] $\delta = 2^{-53}$
0011111111110000000000000000000000000000000000000000000000000000
0011111111110000000000000000000000000000000000000000000000000001
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000010

[2, 4] $\delta = 2^{-51}$
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000100
0011111111110000000000000000000000000000000000000000000000000110
0011111111110000000000000000000000000000000000000000000000001000
0011111111110000000000000000000000000000000000000000000000001010

3.4 Wnioski

[1,2] Reprezentacja bitowa dla Float64 dla kolejnych liczb rózni się o 01, zatem każdy kolejny krok jest poprawny. Każda następna powstaje przez dodanie 01 do poprzedniej.

Jak widać w kolejnych przedziałach będaych potęgami dwójki są zawsze rozmieszczone z krokiem 2^-n im większy przedział tym większe n. Zauważalne jest też, że kolejne liczby między potęgami dwójki maja tą samą cechę, zmienia się natomiast mantysa.

- ## *Zadanie 4*

4.1 Opis problemu

Należy znaleźć najmniejszą liczbę x w arytmetyce Float64, taką że x zawiera się w (1,2) i x * 1/x !=1

4.2 Rozwiązanie problemu

Pseudokod algorytmu wyznaczenia szukanej wartości.

x ← 1.0
while (x<2.0)

```
        while( x * (1/x)) != 1.0)
        return x
        end
    x ← nextfloat(x)
    end
```

4.3 Wyniki

x = 1.000000057228997

4.4 Wnioski

Działania w arytmetyce zmiennoprzecnkowej mogą generować błędy. Wynikają one z niedokładności zaokrągleń. Wynik jest sprzecznością, ponieważ w zbiorze liczb rzeczywistych równanie nie ma rozwiązań.

- *Zadanie 5*

5.1 Opis problemu

Wyznaczenie iloczynu skalarnego wektorów:
x=[2.718281828,−3.141592654,1.414213562,0.5772156649,0.3010299957]
y= [1486.2497,878366.9879,−22.37492,4773714.647,0.000185049]
na podstawie 4 podanych algorytmów.

5.2 Rozwiązanie problemu

Wykosztystując algorytmy:
(a) “w przód”
(b) “w tył”
(c) od największego do najmniejszego (dodaj dodatnie liczby w porządku od największegodo najmniejszego, dodaj ujemne liczby w porządku od najmniejszego do największego,a następnie daj do siebie obliczone sumy częściowe)
(d) od najmniejszego do największego (przeciwnie do metody (c))
Obliczamy wynik i porównujemy go z pozostałymi metodami.

5.3 Wyniki

Algorytm	a	b	c	d
Float32	-0.4999443	-0.4543457	-0.5	-0.5
Float64	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0

5.4 Wnioski

Poprawna wartość to $-1.00657107000000 \cdot 10^{-11}$. Jak widać żadna z metod nie uzyskała poprawnego wyniku. Wynik jest najbliższy prawdy dla precyzji Float64 algorytmu a.

- ## *Zadanie 6*

6.1 Opis problemu

Policzenie wartości w arytmetyce Float64 wartości funkcji f i g :

f(x) = (sqrt(x*x+1) -1) , g(x) = x*x / (sqrt(x*x+1) +1)

6.2 Rozwiązanie problemu

Mimo, iż f=g to komputer daje różne wartości. Liczymy wartości dla x=8^-1,8^-2 ...

6.3 Wyniki

f(x)	g(x)
0.0077822185373186414	0.0077822185373187065
0.00012206286282867573	0.00012206286282875901
1.9073468138230965e-6	1.907346813826566e-6
2.9802321943606103e-8	2.9802321943606116e-8
4.656612873077393e-10	4.6566128719931904e-10
7.275957614183426e-12	7.275957614156956e-12
1.1368683772161603e-13	1.1368683772160957e-13
1.7763568394002505e-15	1.7763568394002489e-15
0.0	2.7755575615628914e-17
0.0	4.336808689942018e-19

6.4 Wnioski

Funkcja g jest bardziej wiarygodna niż funkcja f. Obie funkcje zmierzają ku 0, natomiast nie powinny go nigdy osiądać. W przypadku funkcji f dzieje się to bardzo szybko. Dlaczego? Prawdopodobnie wartość całego pierwiastka jest zaookrąglana do 1 i w wyniku odejmowania -1 staje się zerem. W funkcji f odejmowanie zastąpione jest dodawaniem. Dlatego precyzyjność funkcji jest na wiele wyższym poziomie

- ## *Zadanie 7*

7.1 Opis problemu

Obliczyć wartość pochodnej funkcji :

$$f(x) = \sin(x) + \cos(3x)$$

, w punkcie $x_0 = 1$, korzystając z wozru:

$$fl(f'(x_0) = (f(x_0+h) - f(x_0)) / h$$

oraz błędu |DERIVATIVE(x)-DERIVATIVE(fl(x))| dla $h = 2^{-n}$, gdzie $n = 0,\dots,54$

7.2 Rozwiązanie problemu

Obliczamy przybliżone wartości pochodnych w $x_0=1$ gdzie

$$f'(x) = \cos(x) – \sin(3*x)$$

błędy dla kolejnych p oraz wartość h+1.

7.3 Wyniki

p	DERIVATIVE	|DERIVATIVE(x)-DERIVATIVE(fl(x))|	h+1
0	2.0179892252685967	1.9010469435800585	2.0
1	1.8704413979316472	1.753499116243109	1.5
2	1.1077870952342974	0.9908448135457593	1.25
3	0.6232412792975817	0.5062989976090435	1.125
4	0.3704000662035192	0.253457784514981	1.0625
5	0.24344307439754687	0.1265007927090087	1.03125
6	0.18009756330732785	0.0631552816187897	1.015625
7	0.1484913953710958	0.03154911368255764	1.0078125
8	0.1327091142805159	0.015766832591977753	1.00390625
. . .	. . .	. . .	. . .
28	0.11694228649139404	4.802855890773117e-9	1.0000000037252903
. . .	. . .	. . .	. . .
47	0.109375	0.007567281688538152	1.000000000000007
48	0.09375	0.023192281688538152	1.0000000000000018
49	0.125	0.008057718311461848	1.0000000000000018
50	0.0	0.11694228168853815	1.0000000000000009
51	0.0	0.11694228168853815	1.0000000000000004
52	-0.5	0.6169422816885382	1.0000000000000002
53	0.0	0.11694228168853815	1.0
54	0.0	0.11694228168853815	1.0

7.4 Wnioski

Po skrupulatnym przyjrzeniu się tabeli możemy stwierdzić, że dla wartości p = 28 błąd jest najmniejszy, a tym samym wynik jest najbardziej precyzyjny. W kolejnych krokach h+1 dąży dalej do 1, a błąd spowrotem wzrasta. Możemy również zauważyć, że dla p = 54 i p =53 wartość przybliżonej pochodznej wynosi 0 wynika to z tego iż h staje się tak małe, że jest pochłaniane przez 1.0. W wyniku czego od pewnego momentu zaczynami się oddalać od poprawnego wyniku.