Obliczenia naukowe
Sprawozdanie – Lista 1

Bartosz Rzepkowski

18.10.2015

1. Zadanie 1

1.1 Opis problemu

Wyznaczyć iteracyjnie zero maszynowe (macheps), precyzję arytmetyki (eta) oraz MAX dla typów zmiennoprzecinkowych standardu IEEE 754.

1.2 Rozwiązanie

Zero maszynowe

```
macheps = 1.0
while 1.0 + macheps / 2.0 > 1.0 do
        macheps = macheps / 2.0
end while
```

Precyzja arytmetyki

```
eta = 1.0
while eta > 0.0 do
        if eta / 2.0 == 0.0
                break
        else
                eta = eta / 2.0
        end if
end while
```

MAX

```
max = 2.0
while isinf(max) = false
        if isinf(max * 2) = true
                break
        else
                max = max * 2
        end if
end while
```

1.3 Wyniki

Typ zmiennopozycyjny	Epsilon wyznaczony iteracyjnie	Wynik metody eps()	Liczba eta wyznaczone iteracyjnie	Wynik metody nexfloat()	MAX wyznaczony iteracyjnie	Wynik metody realmax()
Float16	0.00097656	0.00097656	5.9605e-8	5.9605e-8	65504.0	65504.0
Float32	1.1920929e-7	1.1920929e-7	1.0e-45	1.0e-45	3.4028235e38	3.4028235e38
Float64	2.220446049250313e-16	2.220446049250313e-16	5.0e-324	5.0e-324	1.7976931348623157e308	1.7976931348623157e308

W języku C wynik metody analogicznej do eps() daje wyniki:

Dla Float32: 1.19209289550781250000e-07

Dla Float64: 2.22044604925031308085e-16

2. Zadanie 2

2.1 Opis problemu

Sprawdzić eksperymentalnie w języku Julia poprawność twierdzenia Kahana, które głosi, że epsilon maszynowy można wyznaczyć za pomocą wzoru

$$3\left(\frac{4}{3}-1\right)-1$$

2.2 Rozwiązanie

Zastosowanie powyższego wzoru dla typów zmiennopozycyjnych standardu IEEE 754

2.3 Wyniki

Typ zmiennopozycyjny	Epsilon maszynowy
Float16	-0.00097656
Float32	1.1920929e-7
Float64	-2.220446049250313e-16

2.4 Wnioski

Komputer nie jest w stanie dokładnie przechowywać liczb rzeczywistych (w tym przypadku liczby 4/3), ponieważ przechowuje je w systemie dwójkowym i z tego powodu otrzymane wyniki musi zaokrąglić z pewną dokładnością. To doprowadza do niedokładnego obliczenia epsilonu maszynowego.

3. Zadanie 3

3.1 Opis problemu

Sprawdzić eksperymentalnie w języku Julia, że w arytmetyce Float64 liczby zmiennopozycyjne są równomiernie rozmieszczone w danych zakresach.

3.2 Rozwiązanie

```
// Dla danego zakresu [a, b] stosujemy poniższy algorytm
delta = nextfloat(a) – a     // Odstęp między pierwszą liczbą zakresu, a jej nastepnikiem
k = 1.0
x = a

while x <= b
        x = x + k * delta
        if x – prevfloat(x) != delta
                println(„Delta różna od poczatkowej”)
        end if
        k = k + 1
end while
```

3.3 Wyniki

Dla podanych niżej przedziałów otrzymałem następujące rozmieszczenie:

Zakres	Krok w zakresie
[½, 1]	2^{-53}
[1, 2]	2^{-52}
[2, 4]	2^{-51}

Algorytm nie wykazał, by w danych przedziałach występowały inne przedziały niż te przyjęte na początku.

3.4 Wnioski

W arytmetyce zmiennopozycyjnej Float64 większe liczby zapisywane są z mniejszą precyzją niż mniejsze.

4. Zadanie 4

4.1 Opis problemu

Znaleźć eksperymentalnie w arytmetyce Float64 liczbę zmiennopozycyjną x w przedziale 1 < x < 2 taką, że

$$x*\left(\frac{1}{x}\right)\neq 1 \quad \text{tj.} \quad fl\left(x\, fl\left(\frac{1}{x}\right)\right)\neq 1 \quad .$$

Następnie znaleźć najmniejszą liczbę spełniającą powyższe równanie.

4.2 Rozwiązanie

W celu rozwiązania zadania zastosowałem poniższy algorytm:

```
1. x = nextfloat(1.0)
2. eta = nextfloat(1.0) – 1.0
3.
4. while fl(x * fl(1/x)) == 1 do
5.         x = x + eta
6. end while
```

W celu obliczenia najmniejszej liczby spełniającej równanie dane w zadaniu na początku algorytmu należy pod x podstawić najmniejszą liczbę dla danego typu zmiennopozycyjnego, która nie jest liczbą nieskończoną.

4.3 Wyniki

A) x = 1.000000057228997

B) x = -1.7976931348623157e308

5. Zadanie 5

5.1 Opis problemu

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]

A) „W przód” - sumując otrzymane iloczyny po kolei od pierwszego do ostatniego

B) „W tył” = sumują otrzymane iloczyny od ostatniego do pierwszego

C) „Od największej do najmniejszej” - obliczyć sumy częściowe – zsumować liczby dodatnie od największej do najmniejszej oraz liczby ujemne od najmniejszej do największej, po czym dodać do

siebie otrzymane sumy częściowe.
D) „Od najmniejszej do największej” - sposób odwrotny do podpunktu C)

5.2 Rozwiązanie

Początkowo dla każdego podpunktu stworzyłem tablicę przechowującą wyniki częściowe:
[4040.045551380452, -2.7594712767027467e6, -31.64291531266504, 2.7554628740109736e6, 5.57052996742893e-5]

W algorytmach A oraz B dodałem elementy tablico odpowiednio od pierwszego do ostatniego i od ostatniego do pierwszego.

W algorytmie C posortowałem otrzymaną tablicę, po czym dodałem do siebie elementy dodatnie (od końca tablicy do początku) oraz elementy ujemne (od początku tablicy do jej końca).

W algorytmie D postąpiłem analogicznie do podpunktu C, jednak sumując elementy w odwrotnej kolejności.

5.3 Wyniki

Typ zmiennopozycyjny	„W przód”	„W tył”	„Od największej do najmniejszej”	„Od najmniejszej do największej”
Float32	-0.4999443	-0.4543457	-0.5	-0.5
Float64	1.0251881368296672e-10	-1.5643308870494366e-10	0.0	0.0

5.4 Wnioski

Najdokładniejszy jest sposób „w tył” dla Float64 (najbliższy rzeczywistej wartości wyrażenia = -1.00657107000000$_{10}$-11). Przy dodawaniu cyfr od największej do największej lub odwrotnie występuje zjawisko pochłonięcia mniejszej liczby przez większą.

6. Zadanie 6

6.1 Opis problemu

Policzyć w języku Julia w arytmetyce Float64 wartości następujących funkcji:

$$f(x)=\sqrt{x^2+1}-1$$

$$g(x)=\frac{x^2}{\sqrt{x^2+1}+1}$$

dla $x=8^{-1},8^{-2},8^{-3},\ldots$

6.2 Rozwiązanie

W celu rozwiązania zadania utworzyłem dwie funkcje, które później użyłem w pętli for:

```
Function f(x)
        fl(sqrt(x^2.0 + 1.0) – 1.0)
end

function g(x)
        fl(x^2.0 / (sqrt(x^2.0 + 1.0) + 1.0))
end

for i = 1 : 179 do
        x = f(8.0 ^ -i)
        y = g(8.0 ^ -i)
end for
```

6.3 Wyniki

```
i = 1.0 | f(x) = 0.0077822185373186414 | g(x) = 0.0077822185373187065
i = 2.0 | f(x) = 0.00012206286282867573 | g(x) = 0.00012206286282875901
i = 3.0 | f(x) = 1.9073468138230965e-6 | g(x) = 1.907346813826566e-6
i = 4.0 | f(x) = 2.9802321943606103e-8 | g(x) = 2.9802321943606116e-8
i = 5.0 | f(x) = 4.656612873077393e-10 | g(x) = 4.6566128719931904e-10
i = 6.0 | f(x) = 7.275957614183426e-12 | g(x) = 7.275957614156956e-12
i = 7.0 | f(x) = 1.1368683772161603e-13 | g(x) = 1.1368683772160957e-13
i = 8.0 | f(x) = 1.7763568394002505e-15 | g(x) = 1.7763568394002489e-15
i = 9.0 | f(x) = 0.0 | g(x) = 2.7755575615628914e-17
…
i = 175.0 | f(x) = 0.0 | g(x) = 4.144523e-317
i = 176.0 | f(x) = 0.0 | g(x) = 6.4758e-319
i = 177.0 | f(x) = 0.0 | g(x) = 1.012e-320
i = 178.0 | f(x) = 0.0 | g(x) = 1.6e-322
i = 179.0 | f(x) = 0.0 | g(x) = 0.0
```

6.4 Wnioski

Funkcja f(x) jest mniej dokładna, ponieważ dochodzi w niej do redukcji cyfr znaczących. Możemy to zaobserwować po wynikach działania programu. Funkcja f(x) osiąga wartość 0.0 już dla $x=8^{-9}$, a g(x) osiąga tę wartość dopiero dla $x=8^{-179}$.

7. Zadanie 7

7.1 Opis problemu

Przy pomocy języka Julia obliczyć przybliżoną wartość pochodnej f(x) = sinx + cos3x w punkcie x_0 = 1 oraz

7.2 Rozwiązanie

W celu rozwiązania zadania stworzyłem następującą funkcję:

```
function derivative(f, x, h)
  fl((f (fl (x + h)) - f(x)) / h)
end
```

którą następnie stosowałem dla $h=2^{-n} dla\, n=0,1,2,\ldots,54$

Obliczałem również błąd między otrzymywanymi wartościami, a rzeczywistym wynikiem pochodnej dla tej funkcji (= 0).

7.3 Wyniki

```
~f'(x) = 0.4041753177546187 dla h = 1.0 | 1 + h = 2.0, |f'(x) - ~f'(x)| = 0.4041753177546187
~f'(x) = 0.41911962221161403 dla h = 0.5 | 1 + h = 1.5, |f'(x) - ~f'(x)| = 0.41911962221161403
~f'(x) = 0.41350582027035543 dla h = 0.25 | 1 + h = 1.25, |f'(x) - ~f'(x)| = 0.41350582027035543
~f'(x) = 0.40741408647966004 dla h = 0.125 | 1 + h = 1.125, |f'(x) - ~f'(x)| = 0.40741408647966004
~f'(x) = 0.4035622508550176 dla h = 0.0625 | 1 + h = 1.0625, |f'(x) - ~f'(x)| = 0.4035622508550176
…
~f'(x) = 0.3991823196411133 dla h = 2.3283064365386963e-10
~f'(x) = 0.3991823196411133 dla h = 1.1641532182693481e-10
~f'(x) = 0.3991813659667969 dla h = 5.820766091346741e-11
~f'(x) = 0.3991813659667969 dla h = 2.9103830456733704e-11
...
~f'(x) = 0.25 dla h = 4.440892098500626e-16 | 1 + h = 1.0000000000000004, |f'(x) - ~f'(x)| = 0.25
~f'(x) = 0.5 dla h = 2.220446049250313e-16 | 1 + h = 1.0000000000000002, |f'(x) - ~f'(x)| = 0.5
~f'(x) = 0.0 dla h = 1.1102230246251565e-16 | 1 + h = 1.0, |f'(x) - ~f'(x)| = 0.0
~f'(x) = 0.0 dla h = 5.551115123125783e-17 | 1 + h = 1.0, |f'(x) - ~f'(x)| = 0.0
```

7.4 Wnioski

Możemy zauważyć, że dla h = 2.220446049250313e-16 dochodzi do zaburzenia (różnica między otrzymanym wynikiem a wartością rzeczywistą nagle rośnie).