Obliczenia naukowe. Lista nr 1. Sprawozdanie.

Kacper Szatan nr 236478

October 21, 2019

1 CEL

Celem listy jest zapoznanie się z językiem programowania *JULIA*, oraz z reprezentacją liczb zmiennoprzecinkowych w standardzie **IEEE 754**. Należało wykonać 7 zadań, które miały skonfrontować naszą wiedzę zdobytą na wykładzie dotyczącą reprezentacji liczb oraz sposobu wykonywania działań w komputerze.

2 REALIZACJA

Poniżej zajmiemy się omówieniem realizacji zadań wykonanych na laboratorium. Wnioski wyciągniętę z przeprowadzonych eksperymentów będą podane w następnym punkcie (**3. WNIOSKI**).

2.1 Zadanie 1 (Rozpoznanie arytmetyki)

Całe zadanie testowano w standardach zgodnych z **IEEE 754** (half, single, double). W pierwszej części zadania musimy iteracyjnie wyznaczyć *macheps* [1]. Następnie porównać otrzymane wyniki z wbudowaną funkcją *eps(typ arytmetyki)* oraz z danymi w pliku nagłówkowym *float.h* z dowolnej instalacji języka C.

```
x = Float16(1)
y = Float32(1)
z = Float64(1)
```

Listing 1.1: Zmienne dla całego zadania pierwszego.

[1] Epsilonem maszynowym macheps (ang. machine epsilon) nazywamy najmniejszą liczbę macheps > 0 taką, że fl$(1.0 +$ macheps$) > 1.0$.

```
function machEpsEx(one)
mach_eps = one
#przypisujemy mach_eps jedynke
#w arytmetyce ktora testujemy
checker = mach_eps
while(one + checker > one)
        mach_eps = checker
        checker = checker / 2
        typeof(one)(checker)
        typeof(one)(mach_eps)
end
#return mach_eps #gdyby byl potrzebny macheps to odkomentowac.
# ponizej ladny format drukujacy
str = string("Calculated macheps ", typeof(mach_eps), " = ",
mach_eps, "\neps(", (typeof(one)), ") = ", eps(typeof(one)), "\n")
print(str)
end
```

Listing 1.2: Funkcja pozwalająca obliczyć macheps iteracyjnie.

W drugiej części zadania mamy iteracyjnie wyznaczyć liczbę *eta* [2]. Następnie porównać otrzymane wyniki dla wywołania funkcji wbudowanej *nextfloat(typ reprezentacji(0.0))*.

```
function etaEx(one)
zero = typeof(one)(0.0)
eta = one
checker = eta
while(checker > zero)
        eta = checker
        checker = checker / 2
        typeof(one)(checker)
        typeof(one)(eta)
end
#return eta #gdyby byla potrzebna eta to odkomentowac.
# ponizej ladny format drukujacy
str = string("Calculated eta ", typeof(eta), " = ",
eta, "\nnextFloat(", (typeof(one)), "(0.0)) = ", nextfloat(typeof(zero)(zero
"floatmin(", (typeof(one)), ") = ", floatmin(typeof(zero)), "\n")
print(str)
end
```

Listing 1.3: Funkcja pozwalająca obliczyć eps iteracyjnie. (Analogiczna do funkcji na Listingu 1.2)

[2] eta to liczba, taka że eta > 0.0

Trzecia część zadania polega na iteracyjnym wyznaczeniu MAX. Skorzystano tu z wbudowanej funkcji *isinf(liczba)*, sprawdzającej czy podana liczba jest nieskończonością. Na początku obliczamy maksymalną liczbę, która jest postaci $x = 2^k\, k \epsilon \mathcal{Z}$. Następnie aby obliczyć właściwe maksimum szukamy liczby postaci $MAX = x * (2 - m)$. Gdzie m jest najmniejszą liczbą przy której zaprezentowane wyrażenie nie jest nieskończonością. Otrzymane wyniki należało porównać wywołaniem funkcji systemowej *floatmax(typ reprezentacji)* oraz z danymi zawartymi w pliku nagłówkowym *float.h* dowolnej instalacji języka C.

```
function maxEx(one)
max = one
checker = max
while(!isinf(checker))
        max = checker
        checker = checker * 2
        typeof(one)(checker)
        typeof(one)(max)
end
min = etaEx(one)
while(isinf(max * (2 - min)))
        min = min * 2
end
max = max * (2 - min)
#return max #gdyby byla potrzebna eta to odkomentowac.
# ponizej ladny format drukujacy
str = string("Calculated MAX ", typeof(max), " = ",
max, "\nfloatmax(", (typeof(one)), ") = ", floatmax(typeof(one)), "\n")
print(str)
end
```

Listing 1.4: Funkcja pozwalająca obliczyć MAX iteracyjnie.

2.2 Zadanie 2

Kahan stwierdził, że epsilon maszynowy (macheps) można otrzymać obliczając wyrażenie $3(4/3 - 1) - 1$ w arytmetyce zmiennopozycyjnej. W tym zadaniu eksperymentalnie sprawdzamy poprawność tego twierdzenia, dla wszystkich typów reprezentacji (half, float, double).

```
function machEpsKahan(one)
# Kahan formula 3(4/3-1)-1
tmp1 = typeof(one)(typeof(one)(4) / typeof(one)(3)) # 4 / 3 = tmp1
tmp2 = typeof(one)(tmp1 - typeof(one)(1)) # tmp1 - 1 = tmp2
tmp3 = typeof(one)(typeof(one)(3) * tmp2) # 3 * tmp2 = tmp3
mach_eps = typeof(one)(tmp3 - typeof(one)(1)) # tmp3 - 1 = mach_eps
return mach_eps
end
```

Listing 2.1: Funkcja sprawdzająca poprawność twierdzenia Kahan'a.

2.3 Zadanie 3

W zadaniu trzecim mamy do zbadania rozmieszczenie liczb w artmetyce Float64 w zadanych przedziałach [0.5 , 1], [1 , 2], [2 , 4]. Wywołanie napisanej funkcji z odpowiednimi parametrami pozwala nam zbadać co się dzieje przy przejściu z przedziału do przedziału i jaki jest aktuala odległość między liczbami. Użyto bitstring do porównania delt.

```
x = Float64(0.999999)
y = Float64(1.00001)
x2 = Float64(1.999999)
y2 = Float64(2.00001)
x3 = Float64(3.999999)
y3 = Float64(4.00001)
function distribution(x, y)
delta = Float64(nextfloat(x) - x)
print("Start delta = ", delta, "\n")
while(x < y)
        x = Float64(x + delta)
        if(x + delta == x)
         print("For x = ", x, " delta = ", delta, " is to low.\n",
         "Distribution of floats is now wider.\n",
         "new delta = old delta * 2 === ", delta * 2, "\n")
         print("BITSTRING delty = ", delta, "\n", bitstring(delta), "\n")
         delta = delta * 2
         print("BITSTRING delty = ", delta, "\n", bitstring(delta), "\n")
                break
        end
end
end

distribution(x, y)
distribution(x2, y2)
distribution(x3, y3)
```

Listing 3.1: Funkcja sprawdzająca rozmieszczenie liczb w zadanych przedziałach.

2.4 Zadanie 4

W zadaniu czwartym należało eksperymentalnie w arytmetyce Float64 wyznaczyć liczbę zmiennopozycyjną x w przedziale $1 < x < 2$, taką, że $x*(1/x) \neq 1$. Znaleziona liczba powinna być możliwie jak najmniejsza.

```
x = Float64(1)
function find(x)
while (x < Float64(2))
        if (Float64(x * (Float64(1 / x))) != Float64(1))
        println(x)
        break
        end
x = nextfloat(x)
end
end
```

2.5 Zadanie 5

W zadaniu 5 należy zaimplementować cztery algorytmy dla podanych danych:

$$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$$

$$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$$

- "w przód" $\sum_{i=1}^{n=5} x_i y_i$

```
function sumA(x, y, n)
s = typeof(x[1])(0)
for i = 1: n
        s = s + typeof(x[1])(x[i] * y[i])
        s = typeof(x[1])(s)
end
return s
end
```

Listing 5.1: funkcja obliczająca sumę w przód

- "w tył" $\sum_{i=n=5}^{1} x_i y_i$ Kod sumy w tył nie został zamieszczony gdyż jest analogiczny i prawie identyczny do sumy w przód.
- od największego do najmniejszego (dodaj dodatnie liczby w porządku od największego do najmniejszego, dodaj ujemne liczby w porządku od najmniejszego do największego, a następnie daj do siebie obliczone sumy częściowe).
- od najmniejszego do największego (przeciwnie do metody (c)).

```
function sumD(x, y, n)
t = sort(newTab(x, y, n))
sum_positive = typeof(x[1])(0)
sum_negative = typeof(x[1])(0)
for i = 1 : n
        if i > 2 # DWIE PIWERWSZE LICZBY SA UJEMNE
                for j = i - 1 :-1: 1
                 sum_negative = typeof(x[1])(sum_negative + t[j])
                end
                for k = i : n
                 sum_positive = typeof(x[1])(sum_positive + t[k])
                end
                break
        end
end
return typeof(x[1])(sum_negative + sum_positive)
end
```

Listing 5.2: Funkcja pozwalająca obliczyć sumę w zaczynając od najbliższych zera wartości. Obliczono sumy częściowe ujemną i dodatnią odzdzielnie i na końcu je dodano.

Wyniki należało porównać z prawidłową wartością (dokładność do 15 cyfr) $-1.00657107000000 * 10^{-11}$.

2.6 Zadanie 6

Zadanie szóste Policz polega na policzeniu w arytmetyce Float64 wartości następujących funckji $f(x) = \sqrt{x^2+1} - 1 \quad g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$, dla $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$. Chociaż $f = g$ komputer daje różne wyniki. Trzeba sprawdzić, które z nich są wiarygodne, a które nie.

```
function f(x)
 return Float64(Float64(sqrt(Float64(square(Float64(x)))+1.0))-1.0)
end
function g(x)
 return Float64(Float64(square(Float64(x))) /
 Float64((Float64(sqrt(Float64(square(Float64(x)))+1.0))+1.0)))
end
function z6()
for i=1:20
        println("f(8^(-", i, ")) = ", f(8.0^(-i)))
        println("g(8^(-", i, ")) = ", g(8.0^(-i)))
end
end
```

Listing 6.1: Funkcja pozwalająca obliczyć wynik f(x) i g(x) i porównać wyniki.

2.7 Zadanie 7

W ostatnim zadaniu należało obliczyć przybliżoną wartość pochodnej $f(x)$ w punkcie x ze wzoru:

$$f_0^{'}(x_0) \approx \tilde{f}_0^{'}(x_0) = \frac{f(x_0 + h) + -f(x_0)}{h}$$

Dla funkcji $f(x) = sin(x) + cos(3x)$ w punkcie $x_0 = 1$.
Obliczyć także błąd $|f_0^{'}(x_0) - \tilde{f}_0^{'}(x_0)|$.
Przeprowadzić badanie dla $h = 2^{-n}$ $(n = 0, 1, 2, 3, ...54)$

```
function f(x)
        return Float64(sin(x) + cos(3.0 * x))
end
function derivativeReal(x)
        return Float64(cos(x) - 3.0 * sin(3.0 * x))
end
function derivativeAprox(x,h)
        return Float64((Float64(f(Float64(x+h)) - f(x))) / Float64(h))
end
function error()
for n = 0:54
 h = Float64(2^(Float64(-n)))
 println("Error for h=2^(-", n, ") = ",
 abs(derivativeReal(1) - derivativeAprox(1, h)))
end
end
function printDeriv()
        for n = 0:54
         h = Float64(2^(Float64(-n)))
         println("Derivative approximation for h=2^(-", n, ") = ",
         derivativeAprox(1, h))
        end
end
```

Listing 7.1: Zestaw funkcji pozwalających na wyliczenie przybliżenia pochodnej w punkcie jak i jej dokładnej wartości. Funkcja error() potrafi podać błąd jaki został popełniony podczas przybliżenia.

3 WYNIKI I WNIOSKI

3.1 Zadanie 1

Z tabelki 1 możemy odczytać, że wraz ze wzrostem precyzji *macheps* maleje. Precyzja arytmetyki którą określaliśmy epsilonem to właściwie macheps/2. MIN_{SUB} to faktyczne najmniejsza liczba jaką możemy zapisać w komputerze. Jest ono postaci $2^{-(t-1)}2^{C_{min}}$. MIN_{NOR} to minimum znormalizowane postaci $2^{C_{min}}$.

Precyzja	macheps	eps()	float.h (macheps)	eta	nextfloat(0.0)	floatmin()
Float16	0.000977	0.000977	-	6.0e-8	6.0e-8	6.104e-5
Float32	1.192093e-7	1.192093e-7	1.192093e-7F	1.0e-45	1.0e-45	1.175e-38
Float64	2.220446e-16	2.220446e-16	2.220446e-16L	5.0e-324	5.0e-324	2.225e-308

Table 1: Wyniki z zadania pierwszego. Widać, że eta odpowiada MIN_{SUB}, a floatmin() MIN_{NOR}.

3.2 Zadanie 2

Precyzja	macheps	Kahan
Float16	0.000977	-0.000977
Float32	1.192093e-7	1.192093e-7
Float64	2.220446e-16	-2.220446e-16

Table 2: Wyniki wyliczeń macheps metodą Kahan'a.

Wyniki są równe co do wartości bezwzględnej. W precyzji Float16 i Float64 otrzymujemy wynik o przeciwnym znaku.

3.3 Zadanie 3

3.4 Zadanie 4

3.5 Zadanie 5

3.6 Zadanie 6

3.7 Zadanie 7

Jak wytłumaczyć, że od pewnego momentu zmniejszanie wartości h nie poprawia przybliżenia wartości pochodnej? Jak zachowują się wartości 1+h? Obliczone przybliżenia pochodnej porównać z dokładną wartością pochodnej, tj. zwróć uwagę na błędy |f