Obliczenia Naukowe Lista1

Bartłomiej Puchała

February 24, 2024

1 Zadanie 1

1.1 Opis problemu

Zadanie polega na napisaniu w Julii programów wyznaczajacych epsilony maszynowe, liczby maszynowe eta oraz liczbe (MAX) dla wszystkich typów zmiennopozycyjnych oraz wywnioskowaniu i odpowiedzeniu na kilka pytań dotyczacych otrzymanych wyników.

1.2 Rozwiazanie

```
function machine_epsilon(T)
    value = one(T)
    epsilon = value
    while value + epsilon > value
        if value + (epsilon / 2) <= value
            break
        else
            epsilon /= 2
        end
    end
    return epsilon
end

function calc_eta(T)
    eta = one(T)
    while eta > 0 && eta / 2 > 0
        eta /= 2
    end
    return eta
end
```

```
function calc_max(T)
    max = one(T)
    while !isinf(max * 2)
        max *= 2
    end
    gap = max / 2
    while !isinf(max + gap) && gap >= one(T)
        max += gap
        gap /= 2
    end
    return max
end
```

1.3 Wyniki

Typ Danych	Moja funkcja	Eps	Float.h
Float16	0.000977	0.000977	unavailable
Float32	1.1920929e-7	1.1920929e-7	1.1920928955e-07
Float64	2.220446049250313e-16	2.220446049250313e-16	2.2204460493e-16

Typ Danych	Moja funkcja	nextfloat(0.0)
Float16	6.0e-8	6.0e-8
Float32	1.0e-45	1.0e-45
Float64	5.0e-324	5.0e-324

Typ Danych	Moja funkcja	floatmax(T)
Float16	6.55e4	6.55e4
Float32	3.4028235e38	3.4028235e38
Float64	1.7976931348623157e308	1.7976931348623157e308

1.4 Wnioski

Zwiazek liczby macheps z precyzja arytmetyki:
Macheps to najmniejsza liczba dodatnia, która można dodać do 1 w danej precyzji arytmetycznej i uzyskać wynik > 1. Zwiazek miedzy machepsem a precyzja arytmetyki można wyrazić za pomoca wzoru

$$(Macheps = 2^{-p})$$

Macheps to wartość machine epsilon dla danej precyzji arytmetyki
p to liczba bitów używanych do reprezentacji liczby zmiennoprzecinkowej(precyzja)
Wnioskiem z tego jest fakt, że im mniejsza wartość macheps, tym wieksza jest precyzja arytmetyki i tym dokładniejsze obliczenia numeryczne można wykonać w danej reprezentacji.

Zwiazek liczby eta z liczba MIN_{sub}:
MIN_{sub} reprezentuje najmniejsza liczbe > 0, która może być reprezentowana w danej precyzji, tym samym jest liczba eta opisana w zadaniu.

Funkcje floatmin(Float32) i floatmin(Float64) oraz zwiazek zwracanych wartości z liczba MIN_{nor}
Funkcja floatmin() zwraca najmniejsza dodatnia liczbe zmiennoprzecinkowa, która może być przedstawiana w danej precyzji arytmetyki w postaci znormalizowanej. Dokładnie tym samym jest liczba MIN_{nor}. Postać znormalizowana oznacza, że pierwszy bit mantysy jest równy 1, a reszta bitów mantysy i cechy jest wykorzystana do reprezentacji jej wartości.

2 Zadanie 2

2.1 Opis zadania

Problem polega na sprawdzeniu, czy macheps można otrzymać obliczajac wyrażenie w arytmetyce zmiennopozycyjnej:

$$3\,(4/3 - 1) - 1$$

2.2 Rozwiazanie

```
function calc_macheps(T)
    value = one(T)
```

```
    value_3 = 3 * value
    value_4 = 4 * value
    macheps = value_3 * (value_4 / value_3 - value) - value
    return macheps
end
```

2.3 Wyniki oraz interpretacja

Typ Danych	Moja funkcja	eps
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Jeżeli nałożymy wartość bezwzgledna na otrzymane wyniki to otrzymamy identyczne wartości jak w funkcji *eps*. Zmiana znaku otrzymywanych wyników wynika z faktu, że w formatach Float16, Float32 i Float64 mantysa jest zapisywana na różnej liczbie bitów, tzn dla Float16 jest to 10bitów znaczacych, dla Float32 23 bity znaczace, a dla Float64 52 bity znaczace. Dla typów Float16 i Float64 ostatnia cyfra mantysy jest 0, a dla typu Float32 1, co decyduje o minusie.

2.4 Wnioski

Majac do czynienia z arytmetyka zmiennopozycyjna trzeba uważać na dokładność reprezentacji, aby nie otrzymać innych wyników niż w normalnej arytmetyce.

3 Zadanie 3

3.1 Opis problemu

Problem polega na sprawdzeniu, czy w arytmetyce Float64 liczby zmiennopozycyjne sa równomiernie rozmieszczone w $[1, 2]$ z krokiem $\delta = 2^{-52}$. Należy sprawdzić, czy każda liczba zmiennopozycyjna x pomiedzy 1 i 2 może być przedstawiona w postaci

$$x = 1 + k\delta$$

, gdzie $k = 1, 2 \ldots, 2^{52} - 1$. Należy również sprawdzić jak sa rozmieszczone liczby w przedziale $[1/2, 1]$ i $[2, 4]$.

3.2 Rozwiazanie

```
function test(a :: Float64, b :: Float64, delta :: Float64)
    last = prevfloat(b) # najwieksza liczba mniejsza od końca przedzialu b
    cecha_first = SubString(bitstring(a), 2:12)
    cecha_last = SubString(bitstring(last), 2:12)
    if cecha_first != cecha_last
        return false
    end
    wykladnik = parse(Int, cecha_first, base = 2)
    if ((2.0^(wykladnik - 1023))*2.0^(-52) != delta)
        return false
    end
    return true
end
check_0_5_1 = test(0.5, 1.0, 2^(-53))
check_1_2 = test(1.0, 2.0, 2^(-52))
check_2_4 = test(2.0, 4.0, 2^(-51))
println(check_1_2)
println(check_0_5_1)
println(check_2_4)

function testInterval(a, delta)::Bool
    tmp = a
    for k in 1:2^(52)-1
        x = a + k * delta
        if(bitstring(x) == bitstring(nextfloat(tmp)))
            tmp += delta
        else
            return false
        end
    end
    return true
end
check_first = testInterval(0.5, 2^(-53))
check_second = testInterval(1.0, 2^(-52))
check_third = testInterval(2.0, 2^(-51))
```

3.3 Wyniki i interpretacja

Z przeprowadzonego eksperymentu wynika, że liczby zmiennopozycyjne w przedziałach sa rozmieszczone nastepujaco:

1. $\left[\frac{1}{2}, 1\right] -> \delta = 2^{-53}$, $x = \frac{1}{2} + k\delta$, gdzie $k = 1, 2, \ldots, 2^{52} - 1$
2. $[1, 2] -> \delta = 2^{-52}$, $x = 1 + k\delta$, gdzie $k = 1, 2, \ldots, 2^{52} - 1$
3. $[2, 4] -> \delta = 2^{-51}$, $x = 2 + k\delta$, gdzie $k = 1, 2, \ldots, 2^{52} - 1$

W moim podejściu w funkcji $test()$ staram sie porównać cechy dwóch skrajnych liczb z przedziału, jeżeli sa one różne to nie możemy mieć równomiernego rozmieszczenia w tym przedziale. Jeżeli sa równe to możemy sprawdzić jak zmienia sie liczba przy powiekszeniu mantysy o jeden. Powyższe rezultaty dotyczace rozmieszczenia liczb w przedziałach uzyskałem dzieki funkcji $testInterval()$.

3.4 Wnioski

Znajac podstawy arytmetyki zmiennopozycyjnej w danej precyzji można wyznaczać, jak sa rozmieszczone liczby w danych przedziałach.

4 Zadanie 4

4.1 Opis problemu

Problem polega na eksperymentalnym znalezieniu w arytmetyce Float64 zgodnej ze standardem IEE 754 liczby zmiennopozycyjnej x przedziale $1 < x < 2$, takiej że

$$x * \frac{1}{x} \neq 1$$

4.2 Rozwiazanie

```
function findNumber(T, a, b)
    curr_x = a
    while curr_x != b
        curr_x += machine_epsilon(Float64)
        reverse_x = 1 / curr_x
        if (curr_x * reverse_x) != 1
            return curr_x
        end
```

```
    end
    println("The end of function")
end

function findNumber_2(T, a, b)
    curr_x = a
    while curr_x != b
        curr_x = nextfloat(curr_x)
        reverse_x = 1 / curr_x
        if (curr_x * reverse_x) != 1
            return curr_x
        end
    end
    println("The end of function")
end
```

4.3 Wyniki i interpretacja

Najmniejsza liczba zmiennopozycyjna spełniajaca to równanie, jaka zwrócił algorytm jest liczba $x = 1.000000057228997$, spełnia ona zdefiniowane w zadaniu równanie:

$$x * \frac{1}{x} = 0.9999999999999999$$

4.4 Wnioski

W arytmetyce liczb zmiennopozycyjnych ze wzgledu na skończona dokładność wyniki niektórych działań moga być niedokładne i niepoprawne.

5 Zadanie 5

5.1 Opis problemu

Problem polega na obliczeniu iloczynu skalarnego dwóch wektorów na 4 różne sposoby:

1. Algorytm "w przód"
2. Algorytm "w tył"
3. Od najwiekszego do najmniejszego
4. Od najmniejszego do najwiekszego

5.2 Rozwiazanie

1. Algorytm "w przód"

```
function calc_forwards(T, vecX, vecY, n)
    sum = one(T) - 1
    for i in 1:n
        sum += vecX[i] * vecY[i]
    end
    return sum
end
```

2. Algorytm "w tył"

```
function calc_backwards(T, vecX, vecY, n)
    sum = one(T) - 1
    curr = n
    while curr >= 1
        sum += vecX[curr] * vecY[curr]
        if curr - 1 >= 1
            curr -= 1
        else
            break
        end
    end
    return sum
end
```

3. Algorytm od najwiekszego do najmniejszego

```
function calc_descending(T, vecX, vecY)
    sum_positive = one(T) - 1
    sum_negative = one(T) - 1
    sum = one(T) - 1
    tmp_positive = T[]
    tmp_negative = T[]
    if length(vecX) == length(vecY)
        for (i, j) in zip(vecX, vecY)
            if i * j > 0
                push!(tmp_positive, i * j)
            else
```

```
                push!(tmp_negative, i * j)
            end
        end
        sort!(tmp_positive, rev = true)
        sort!(tmp_negative)
        for i in tmp_positive
            sum_positive += i
        end
        for j in tmp_negative
            sum_negative += j
        end
        sum = sum_positive + sum_negative
        println(sum)
    else
        println("Wektory nie sa prawidlowe")
    end
end
```

4. Algorytm od najmniejszego do najwiekszego

```
function calc_ascending(T, vecX, vecY)
    sum_positive = one(T) - 1
    sum_negative = one(T) - 1
    sum = one(T) - 1
    tmp_positive = T[]
    tmp_negative = T[]
    if length(vecX) == length(vecY)
        for (i, j) in zip(vecX, vecY)
            if i * j > 0
                push!(tmp_positive, i * j)
            else
                push!(tmp_negative, i * j)
            end
        end
        sort!(tmp_positive)
        sort!(tmp_negative, rev = true)
        for i in tmp_positive
            sum_positive += i
        end
        for j in tmp_negative
            sum_negative += j
        end
        sum = sum_positive + sum_negative
        println(sum)
    end
end
```

5.3 Wyniki oraz interpretacja

Algorytm	Float32	Float64
W Przód	-0.4999443	1.0251881368296672e-10
W Tył	-0.4543457	-1.5643308870494366e-10
Od Najw.	-0.5	0.0
Od Najm.	-0.5	0.0

Prawidłowy iloczyn obu wektorów wynosi $-1.00657107000000 * 10^{-11}$, przybliżony wynik otrzymujemy w przypadku algorytmu "w tył" dla podwójnej

precyzji(Float64). Dla pojedynczej precyji(Float32) wyniki sa najbardziej rozbieżne z prawidłowym wynikiem, w przypadku Float64 przybliżony wynik otrzymujemy tylko w przypadku algorytmu "w tył".

5.4 Wnioski

Wnioskiem z otrzymanych wyników jest fakt, że kolejność wykonywania działań w arytmetyce zmiennopozycyjnej jest bardzo ważna dla otrzymywanych wyników. W przypadku dodawnia małej i dużej liczby istnieje ryzyko utraty cyfr znaczacych i w zwiazku z tym ryzyko otrzymania niepoprawnego wyniku. Należy zatem zwracać uwage na kolejność wykonywania działań podczas projektowania algorytmów wykorzystywanych do obliczeń numerycznych.

6 Zadanie 6

6.1 Opis problemu

Problem polega na policzeniu w arytmetyce Float64 wartości nastepujacych funkcji dla kolejnych argumentów $x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$.

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

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

6.2 Rozwiazanie

```
function calc_function(x, func, arr)
    for i in -1:-1:-15
        value = func(x^(i))
        push!(arr, value)
    end
end

function calc_f(x)
    return sqrt(x^2 + 1) -1
end

function calc_g(x)
    return x^2 / (sqrt(x^2 + 1) + 1)
end
```

```
arr1_f = Float64[]
calc_function(8.0, calc_f, arr1_f)
println(arr1_f)
arr2_g = Float64[]
calc_function(8.0, calc_g, arr2_g)
println(arr2_g)
```

6.3 Wyniki i innterpretacja

Wartość x	$f(8^{-x})$	$g(8^{-x})$
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
20	0.0	3.76158192263132e-37
50	0.0	2.4545467326488633e-91
100	0.0	1.204959932551442e-181
150	0.0	5.915260930833874e-272
178	0.0	1.6e-322
179	0.0	0.0
180	0.0	0.0
200	0.0	0.0

Z powyższych wyników można wywnioskować, że funkcja g radzi sobie znacznie lepiej z obliczeniem własnych wartości, które sa blisko swoich minimalnych

wartości w zakresie typu danych Float64. Do wartości $x \leq 8$ można zauważyć że funkcje zwracaja bardzo podobne wyniki, lecz nie takie same.

6.4 Wnioski

Wiarygodne sa wyniki zwracane przez funkcje g, poniewaz jezeli mamy dwie liczby $x \approx y$, w tym przypadku $\sqrt{x^2+1} -> 1$ i 1 to odejmujac je od siebie otrzymujemy blad wzgledny(δ) znacznie wiekszy niz precyzja arytmetyki(ϵ), dlatego nalezy unikac odejmowania liczb bliskich sobie w arytmetyce zmiennopozycyjnej i spróbować przekształcić wyrażenie, tak jak zostało przekształcone w funkcji $g(x)$.

$$|\delta| = \frac{|x - y - [rd(x) - rd(y)]|}{|x - y|} \leq \epsilon * \frac{|x| + |y|}{|x - y|}$$

7 Zadanie 7

7.1 Opis problemu

Problem polega na obliczeniu przybliżonej wartości pochodnej funkcji $f(x) = \sin x + \cos 3x$ w punkcie $x_0 = 1$ oraz błedów $\left|f'(x_0) - \hat{f}'(x_0)\right|$ dla $h = 2^{-n}(n = 0, 1, 2, \ldots, 54)$ za pomoca nastepujacego wzoru:

$$f'(x_0) \approx \hat{f}'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h}$$

7.2 Rozwiazanie

```
function approximate_derivative(dArr, precArr, arg, func)
    for n in 0:54
        h = 2.0^(-n)
        approximation = (func(arg + h) - func(arg)) / h
        precision_error = abs(derivative(arg) - approximation)
        push!(dArr, approximation)
        push!(precArr, precision_error)
    end
end

function f(x)
    return sin(x) + cos(3*x)
end
```

```
function derivative(x)
    return cos(x) - 3*sin(3*x)
end
```

7.3 Wyniki oraz interpretacja

h	Approximation	Approximation error
2^{-0}	2.0179892252685967	1.9010469435800585
2^{-1}	1.8704413979316472	1.753499116243109
2^{-2}	1.1077870952342974	0.9908448135457593
2^{-3}	0.6232412792975817	0.5062989976090435
2^{-4}	0.3704000662035192	0.253457784514981
2^{-5}	0.24344307439754687	0.1265007927090087
2^{-6}	0.18009756330732785	0.0631552816187897
2^{-7}	0.1484913953710958	0.03154911368255764
2^{-8}	0.1327091142805159	0.015766832591977753
2^{-9}	0.1248236929407085	0.007881411252170345
2^{-10}	0.12088247681106168	0.0039401951225235265
2^{-11}	0.11891225046883847	0.001969968780300313
2^{-12}	0.11792723373901026	0.0009849520504721099
2^{-13}	0.11743474961076572	0.0004924679222275685
2^{-14}	0.11718851362093119	0.0002462319323930373
2^{-15}	0.11706539714577957	0.00012311545724141837
2^{-16}	0.11700383928837255	6.155759983439424e-5
2^{-17}	0.11697306045971345	3.077877117529937e-5
2^{-18}	0.11695767106721178	1.5389378673624776e-5
2^{-19}	0.11694997636368498	7.694675146829866e-6
2^{-20}	0.11694612901192158	3.8473233834324105e-6
2^{-21}	0.11694420524872848	1.9235601902423127e-6
2^{-22}	0.11694324295967817	9.612711400208696e-7
2^{-23}	0.11694276239722967	4.807086915192826e-7
2^{-24}	0.11694252118468285	2.394961446938737e-7
2^{-25}	0.116942398250103	1.1656156484463054e-7
2^{-26}	0.11694233864545822	5.6956920069239914e-8
2^{-27}	0.11694231629371643	3.460517827846843e-8
2^{-28}	0.11694228649139404	4.802855890773117e-9
2^{-29}	0.11694222688674927	5.480178888461751e-8
2^{-30}	0.11694216728210449	1.1440643366000813e-7
2^{-31}	0.11694216728210449	1.1440643366000813e-7
2^{-32}	0.11694192886352539	3.5282501276157063e-7
2^{-33}	0.11694145202636719	8.296621709646956e-7

2^{-34}	0.11694145202636719	8.296621709646956e-7
2^{-35}	0.11693954467773438	2.7370108037771956e-6
2^{-36}	0.116943359375	1.0776864618478044e-6
2^{-37}	0.1169281005859375	1.4181102600652196e-5
2^{-38}	0.116943359375	1.0776864618478044e-6
2^{-39}	0.11688232421875	5.9957469788152196e-5
2^{-40}	0.1168212890625	0.0001209926260381522
2^{-41}	0.116943359375	1.0776864618478044e-6
2^{-42}	0.11669921875	0.0002430629385381522
2^{-43}	0.1162109375	0.0007313441885381522
2^{-44}	0.1171875	0.0002452183114618478
2^{-45}	0.11328125	0.003661031688538152
2^{-46}	0.109375	0.007567281688538152
2^{-47}	0.109375	0.007567281688538152
2^{-48}	0.09375	0.023192281688538152
2^{-49}	0.125	0.008057718311461848
2^{-50}	0.0	0.11694228168853815
2^{-51}	0.0	0.11694228168853815
2^{-52}	-0.5	0.6169422816885382
2^{-53}	0.0	0.11694228168853815
2^{-54}	0.0	0.11694228168853815

Z otrzymanych wyników można zauważyć, że do $n \leq 28$ przybliżone wartości pochodnej funkcji staja sie bardziej dokładne, a błedy aproksymacji maleja. Najmniejszy bład aproksymacji i najdokładniejsza wartość pochodnej wystepuje dla $n = 28$, bład aproksymacji jest rzedu 10^{-9}, natomiast dla $n > 28$ błedy zaczynaja rosnać, aby na sam koniec wynieść 100%. Dla $1 + h$ błedy aproksymacji sa znacznie wieksze niż dla samego h, wynika to z faktu, że do dużej liczby(1) dodajemy wraz ze wzrostem n coraz mniejsza liczbe.

7.4 Wnioski

Taka niedokładność dla bardzo małych wartości h jest spowodowana faktem, że w przypadku małych liczb zmiennoprzecinkowych mamy mało cyfr znaczacych w zapisie, co powoduje że wraz z ich maleniem tracona jest dokładność i błedy staja sie coraz wieksze. Należy wiec unikać wartości bardzo bliskich zeru.