Obliczenia Naukowe

Karol Janic

21 października 2023

Spis treści

1 Zadanie 1

1.1 Cel

Celem zadania jest wyznaczenie liczb:

- `macheps` - najmniejszej liczby, takiej że $fl(\texttt{1.0 + macheps}) > 1.0$ oraz $fl(\texttt{1.0 + macheps}) = 1.0+$`macheps`
- `eta` - najmniejszej dodatniej liczby
- `max` - największej reprezentowalnej liczby

w arytmetykach 16-, 32- oraz 64-bitowej, porównanie ich wartości z wartościami zwracanymi przez wbudowane funkcje w języku `Julia` oraz wartościami zdefiniowanymi w języku `C`. Dodatkowo należy ocenić związki `macheps` z precyzją arytmetyki, MIN_{sub} z `eta`, MIN_{nor} z wbudowaną funkcją języka `Julia` `floatmin` oraz MAX z `floatmax`. Określone są one przez wzory:

1.2 Rozwiązanie

Wymienione powyżej liczby obliczane są w sposób iteracyjny. Obliczenie wykonywane są w arytmetyce zmiennopozycyjnej:

- `macheps` inicjalizowany jest wartością 1.0 a następnie połowiony do momentu aż dodanie jego połowy do 1.0 nie da w wyniku liczby większej niż 1.0
- `eta` inicjalizowana jest wartością 1.0 a następnie połowiona dopóki jego połowa jest większa od 0.0
- `max` inicjalizowany jest wartością 1.0 a następnie podwajany dopóki jego podwojona wartośc nie będzie nieskończonością. Następnie jest powiększany o coraz mniejsze liczby dopóki dodanie kolejnej liczby nie spowoduje osiągnięcia nieskończoności

Algorytm 1 Wyznaczanie `macheps`

$macheps \leftarrow 1.0$
while $fl(1.0 + \frac{macheps}{2}) > 1.0$ **do**
  $macheps \leftarrow \frac{macheps}{2}$
end while

Algorytm 2 Wyznaczanie `eta`

$eta \leftarrow 1.0$
while $fl(\frac{eta}{2}) > 0.0$ **do**
  $eta \leftarrow \frac{eta}{2}$
end while

Algorytm 3 Wyznaczanie `max`

$max \leftarrow 1.0$
while $fl(2 * max) \neq \infty$ **do**
  $max \leftarrow 2 * max$
end while
$gap \leftarrow \frac{max}{2}$
while $fl(max + gap) \neq \infty$ and $fl(gap) > 0.0$ **do**
  $max \leftarrow max + gap$
  $gap \leftarrow \frac{gap}{2}$
end while

1.3 Wyniki i wnioski

Wartości `macheps` wyznaczone przy użyciu opisanego wyżej algorytmu pokrywają się z wartościami otrzymanymi z wbudowanych funkcji języka `Julia` oraz są bliskie stałymi w języku `C` (za wyjątkiem 16-bitowej reprezentacji, której język `C` nie zapewnia).

typ	wartość wyznaczona	wartość `eps` w `Julii`	wartość w `float.h` w `C`
`Float16`	0.000977	0.000977	niezdefiniowana
`Float32`	1.1920929e-7	1.1920929e-7	1.192093e-7
`Float32`	2.220446049250313e-16	2.220446049250313e-16	2.220446e-16

Tabela 1: Porównanie wartości `macheps`.

Prezycję ayrtmetyki ϵ określa formuła:

$$\epsilon = \frac{1}{2}\beta^{t-1},$$

gdzie β jest bazą rozwinięcia, a t liczbą cyft użytych do zapisu mantysy. Dla typów `Float16`, `Float32`, `Float64` $\beta = 2$ a t przyjmuje kolejno wartości 10, 23, 52. Łatwo można sprawdzić, że wartości `macheps` pokrywają się z prezycją arytmetyki.

typ	`macheps`	ϵ
`Float16`	0.000977	0.000977
`Float32`	1.1920929e-7	1.1920929e-7
`Float64`	2.220446049250313e-16	2.220446049250313e-16

Tabela 2: Porównanie wartości `macheps` z ϵ.

Wartości `eta` wyznaczone przy użyciu opisanego wyżej algorytmu pokrywają się z wartościami otrzymanymi z wbudowanych funkcji języka `Julia`. Język `C` nie definiuje takiej stałej.

typ	wartość wyznaczona	wartość `nextfloat(Typ(0.0))` w `Julii`
`Float16`	6.0e-8	6.0e-8
`Float32`	1.0e-45	1.0e-45
`Float64`	5.0e-324	5.0e-324

Tabela 3: Porównanie wartości `eta`.

MIN_{sub} jest najmiejszą dla arytmetyki nieznormalizowaną liczbą. MIN_{nor} jest najmniejszą dla arytmetyki znormalizowaną liczbą.

$$MIN_{sub} = 2^{1-t} \cdot 2^{c_{min}}$$

$$MIN_{nor} = 2^{c_{min}},$$

gdzie c_{min} jest minimalną możliwą do zapisania cechą wyznaczaną ze wzoru:

$$c_{min} = -2^{d-1} + 2,$$

gdzie d jest liczbą bitów przeznaczonych na zapis cechy. Dla typów zmiennopozycyjnych `Float16`, `Float32`, `Float64` są to kolejno 5, 8, 11.

typ	`eta`	MIN_{sub}	MIN_{nor}	`floatmin(typ)`
`Float16`	6.0e-8	6.0e-8	6.104e-5	6.104e-5
`Float32`	1.0e-45	1.0e-45	1.1754944e-38	1.1754944e-38
`Float64`	5.0e-324	5.0e-324	2.2250738585072014e-308	2.2250738585072014e-308

Tabela 4: Porównanie wartości `eta`, MIN_{sub}, MIN_{nor} oraz `floatmin`.

Z powyższych obliczeń wynika, że wartość MIN_{sub} jest równa `eta` a MIN_{nor} `floatmin`. Można zauważyc, że wartości MIN_{nor} są większe od wartości MIN_{sub}.

Wartości `max` wyznaczone przy użyciu opisanego wyżej algorytmu pokrywają się z wartościami otrzymanymi z wbudowanych funkcji języka `Julia` oraz są bliskie stałymi w języku `C` (za wyjątkiem 16-bitowej reprezentacji, której język `C` nie zapewnia).

typ	wartość wyznaczona	Wartość `floatmax(typ)` w `Julii`	wartość w `float.h` w `C`
`Float16`	6.55e4	6.55e4	niezdefiniowana
`Float32`	3.4028235e38	3.4028235e38	3.402823e38
`Float64`	1.7976931348623157e308	1.7976931348623157e308	1.797693e308

Tabela 5: Porównanie wartości `max`.

2 Zadanie 2

2.1 Cel

Celem zadania jest sprawdzenie poprawności wzoru Kahana na wartość epsilona maszynowego. Przedstawił on następujący wzór:

$$\epsilon_{Kahan} = 3 \cdot (\frac{4}{3} - 1) - 1$$

2.2 Rozwiązanie

Zaimplementowano funkcję wzracającą wartość epsilona przedstawioną przez Kahana. Dla różnych typów danych wygenerowano ją oraz porównano z wartościami `macheps`.

2.3 Wyniki i wnioski

Z przeprowadzonych obliczeń wynika, że wartości obliczone na podstawie wzoru Kahana różnią się wyłącznie znakiem od wartości `macheps`.

typ	`macheps`	ϵ_{Kahan}
`Float16`	0.000977	-0.000977
`Float32`	1.1920929e-7	1.1920929e-7
`Float64`	2.220446049250313e-16	-2.220446049250313e-16

Tabela 6: Porównanie wartości `macheps` z wartościami obliczonymi ze wzoru Kahana.

3 Zadanie 3

3.1 Cel

Celem zadania jest sprawdzenie, czy liczby w arytmetyce zmiennopozycyjnej `Float64` są równomiernie rozłożone w przedziale $[1, 2]$ z krokiem $\delta = 2^{-52}$ oraz w przedziałach $[\frac{1}{2}, 1]$ i $[2, 4]$.

3.2 Rozwiązanie

W celu odpowiedzi na powyższe pytanie w podanych przedziałach generowano kolejne liczby w arytmetyce `Float64` na dwa różne sposoby oraz porównano je. Jednym z nich jest dodawanie δ zaś drugim zwiekszanie liczby reprezentującej mantysę o jeden.

3.3 Wyniki i wnioski

W wyniku działania programu dla przedziału $[1, 2]$ nie wykryto rozbieżności. W przypadku dwóch pozostałych przedziałów oraz takiej samej δ test daje wynik negatywny. Aby sprawdzić co jest tego przyczyną wypisano po kilka wartości liczb i ich bitowych reprezentacji dla przedziałów $[1, 2]$ oraz $[\frac{1}{2}, 1]$. Przedstawiają się one następująco:

wartość	zapis bitowy
0.5000000000000000	0 01111111110 0000000000000000000000000000000000000000000000000000
0.5000000000000001	0 01111111110 0000000000000000000000000000000000000000000000000001
0.5000000000000002	0 01111111110 0000000000000000000000000000000000000000000000000010
0.5000000000000003	0 01111111110 0000000000000000000000000000000000000000000000000011
⋮	⋮
0.9999999999999997	0 01111111110 1111111111111111111111111111111111111111111111111101
0.9999999999999998	0 01111111110 1111111111111111111111111111111111111111111111111110
0.9999999999999999	0 01111111110 1111111111111111111111111111111111111111111111111111
1.0000000000000000	0 01111111111 0000000000000000000000000000000000000000000000000000

Tabela 7: Bitowy zapis kilku liczb z przedziału $[\frac{1}{2}, 1]$.

wartość	zapis bitowy
1.0000000000000000	0 01111111111 0000000000000000000000000000000000000000000000000000
1.0000000000000002	0 01111111111 0000000000000000000000000000000000000000000000000001
1.0000000000000004	0 01111111111 0000000000000000000000000000000000000000000000000010
1.0000000000000007	0 01111111111 0000000000000000000000000000000000000000000000000011
⋮	⋮
1.9999999999999993	0 01111111111 1111111111111111111111111111111111111111111111111101
1.9999999999999996	0 01111111111 1111111111111111111111111111111111111111111111111110
1.9999999999999998	0 01111111111 1111111111111111111111111111111111111111111111111111
2.0000000000000000	0 10000000000 0000000000000000000000000000000000000000000000000000

Tabela 8: Bitowy zapis kilku liczb z przedziału $[1, 2]$.

Można zauważyć, że w przypadku przedziału $[\frac{1}{2}, 1]$ maleje wartość cechy. Bity mantysy zachowują się w taki sam sposób jak w przypadku przedziału $[1, 2]$. Wynika z tego, że wartość δ dla przedziału $[\frac{1}{2}, 1]$ powinna zostać pomniejszona o połowę a w przypadku przedziału $[1, 2]$ podwojona. Dla tak zmienionych delt testy zakończyły się powodzeniem.
Z tak przeprowadzonego doświadczenia można wyciągnąć kilka wniosków:

- liczby w arytmetyce zmiennopozycyjnej są równomiernie rozłożone na przedziałach $[2^k, 2^{k+1}]$
- każdy przedział $[2^k, 2^{k+1}]$ zawiera tyle samo liczb
- gdy potęgi dwójki będące końcami przedziałów rosną, rośnie także δ

4 Zadanie 4

4.1 Cel

Celem zadania jest wyznaczenie najmniejszej liczby x w arytmetyce `Float64` z przedziału $(1, 2)$, takiej, że $x \cdot \frac{1}{x} \neq 1$.

4.2 Rozwiązanie

Z poprzedniego zadania wynika, że w przedziale $(1, 2)$ kolejne liczby różnią się o $\delta = 2^{-52}$. Zatem algorytm, będzie przechodził po kolejnych liczbach zaczynając od $1.0 + \delta$ dopóki nie natrafi na liczbę, której iloczyn z jej odwrotnością w arytmetyce zmiennopozycyjnej nie wyniesie 1.0.

Algorytm 4 Wyznaczanie 'złej' odwrotności

Input: $\delta > 0$
  $x \leftarrow 1.0 + \delta$
  while $fl(x + \delta) \cdot fl(\frac{1}{x+\delta}) = 1.0$ **do**
    $x \leftarrow x + \delta$
  end while

4.3 Wyniki i wnioski

Najmiejszą liczbą spełniającą podane wyżej założenia jest $x = 1.000000057228997$.
Z przeprowadzonego doświadczenia wynika, że w arytmetyce zmiennopozycyjnej nie zawsze zachodzą własności algebraiczne prawdziwe w liczbach rzeczywistych.

5 Zadanie 5

5.1 Cel

Celem zadania jest eksperymentalne porównanie czterech metod obliczania iloczynu skalarnego:

- "w przód"
- "w tył"
- "od największego do najmniejszego"
- "od najmniejszego do największego"

dla wektorów:
$X = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$ oraz
$Y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$

5.2 Rozwiązanie

Zaimplementowano cztery algorytmy przedstawione na poniższych listingach oraz porównano ich działanie dla różnych arytmetyk zmiennopozycyjnych: `Float32`, `Float64`.

Algorytm 5 Wyznaczanie iloczyny skalarnego metodą "w przód"

Input: $X[1 \dots n]$, $Y[1 \dots n]$
  $product \leftarrow 0.0$
  for $i \leftarrow 1$ to n **do**
    $product \leftarrow product + X[i] \cdot Y[i]$
  end for

Algorytm 6 Wyznaczanie iloczyny skalarnego metodą "w tył"

Input: $X[1 \dots n]$, $Y[1 \dots n]$
  $product \leftarrow 0.0$
  for $i \leftarrow n$ to 1 **do**
    $product \leftarrow product + X[i] \cdot Y[i]$
  end for

Algorytm 7 Wyznaczanie iloczyny skalarnego metodą "od największego do najmniejszego"

Input: $X[1 \dots n]$, $Y[1 \dots n]$
  $pos\_partials \leftarrow []$, $neg\_partials \leftarrow []$
  $n_1 \leftarrow 1$, $n_2 \leftarrow 1$
  for $i \leftarrow i$ to n **do**
    $partial \leftarrow X[i] \cdot Y[i]$
    if partial > 0 **then**
      $pos\_partials[n_1] \leftarrow partial$
      $n_1 \leftarrow n_1 + 1$
    else
      $neg\_partials[n_2] \leftarrow partial$
      $n_2 \leftarrow n_2 + 1$
    end if
  end for

  sort $pos\_partials$ **descending**
  sort $neg\_partials$ **ascending**

  $pos\_partial\_product \leftarrow 0.0$
  for $i \leftarrow 1$ to n_1 **do**
    $pos\_partial\_product \leftarrow pos\_partial\_product + pos\_partials[i]$
  end for
  $neg\_partial\_product \leftarrow 0.0$
  for $i \leftarrow 1$ to n_2 **do**
    $neg\_partial\_product \leftarrow neg\_partial\_product + neg\_partials[i]$
  end for
  $product \leftarrow pos\_partial\_product + neg\_partial\_product$

Algorytm 8 Wyznaczanie iloczyny skalarnego metodą "od najmniejszego do największego"

Input: $X[1 \dots n]$, $Y[1 \dots n]$
  $pos\_partials \leftarrow []$, $neg\_partials \leftarrow []$
  $n_1 \leftarrow 1$, $n_2 \leftarrow 1$
  for $i \leftarrow i$ to n **do**
    $partial \leftarrow X[i] \cdot Y[i]$
    if partial > 0 **then**
      $pos\_partials[n_1] \leftarrow partial$
      $n_1 \leftarrow n_1 + 1$
    else
      $neg\_partials[n_2] \leftarrow partial$
      $n_2 \leftarrow n_2 + 1$
    end if
  end for

  sort $pos\_partials$ **ascending**
  sort $neg\_partials$ **descending**

  $pos\_partial\_product \leftarrow 0.0$
  for $i \leftarrow 1$ to n_1 **do**
    $pos\_partial\_product \leftarrow pos\_partial\_product + pos\_partials[i]$
  end for
  $neg\_partial\_product \leftarrow 0.0$
  for $i \leftarrow 1$ to n_2 **do**
    $neg\_partial\_product \leftarrow neg\_partial\_product + neg\_partials[i]$
  end for
  $product \leftarrow pos\_partial\_product + neg\_partial\_product$

5.3 Wyniki i wnioski

Prawidłową wartością jest -1.00657107000000e-11. Otrzymane eksperymentalnie wyniki przedstawiają się następująco:

metoda	typ	
	`Float32`	`Float64`
"w przód"	-0.4999443	1.0251881368296672e-10
"w tył"	-0.454345	-1.5643308870494366e-10
"od największego do najmniejszego"	-0.5	0.0
"od najmniejszego do największego"	-0.5	0.0

Tabela 9: Porównanie wartości iloczynu skalarnego obliczonego różnymi sposobami.

Żadna z metod nie zwraca dokładnej wartości. Obliczenia w arytmetyce `Float32` są bardzo niedokładne.
W przypadku zwiększonej precyzji otrzymane wyniki są bliższe wartości dokładnej. Z doświadczenia wynika, że kolejność wykonywania operacji w arytmetyce zmiennopozycyjnej ma wpływ na wynik końcowy obliczeń. Najlepsze efekty zapewniają metody, które dodają wartości w posortowanej kolejności.
Przyczyną rozbieżności jest fakt, że wektory X i Y są prawie prostopadłe zatem iloczyn jest blisko zeru oraz rzędy wielkości składowych wektorów bardzo się różnią.

6 Zadanie 6

6.1 Cel

Celem zadania jest porównanie wartości dwóch równoważnych funkcji:

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

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

6.2 Rozwiązanie

Zaimplementowano dwie procedury realizujące wyżej podane funkcje. Następnie w arytmetyce `Float64` obliczono ich wartości dla pierwszych dwóstu argumentów w podanej wyżej postaci.

6.3 Wyniki i wnioski

Kilka obliczonych wartości prezentuje się następująco:

x	$f(x)$	$g(x)$
8^{-1}	0.0077822185373186414	0.0077822185373187065
8^{-2}	0.00012206286282867573	0.00012206286282875901
8^{-3}	1.9073468138230965e-6	1.907346813826566e-6
8^{-4}	2.9802321943606103e-8	2.9802321943606116e-8
8^{-5}	4.656612873077393e-10	4.6566128719931904e-10
8^{-6}	7.275957614183426e-12	7.275957614156956e-12
8^{-7}	1.1368683772161603e-13	1.1368683772160957e-13
8^{-8}	1.7763568394002505e-15	1.7763568394002489e-15
8^{-9}	0.0	2.7755575615628914e-17
8^{-10}	0.0	4.336808689942018e-19
$\vdots$	$\vdots$	$\vdots$
8^{-178}	0.0	1.6e-322
8^{-179}	0.0	0.0

Z przeprowadzonego doświadczenia wynika, że pomimo tego, że funkcje są sobie równe w sensie matematycznym, ich wartości dla tych samych argumentów różnią się. Ponadto wartości funkcji f szybko stają się niewiarygodne. Może być to skutkiem odejmowanie od siebie bliskich siebie liczb, które powoduje duży błąd względny.
W przypadku funkcji g wartości stają się bezużyteczne o wiele później. Należy z tego wysnuć wniosek, że przeformułowanie problemu może poprawić dokładność obliczeń zmiennopozycyjnych.

7 Zadanie 7

7.1 Cel

Celem zadania jest wyznaczenie przybliżonej wartości pochodnej funkcji $f(x) = \sin x + \cos 3x$ w punkcie $x_0 = 1$ korzystając ze wzoru $f'(x_0) \approx \frac{f(x_0+h)-f(x)}{h}$, gdzie $h = 2^{-n}$ dla $n = 0, 1, 2, \ldots, 54$. Należy także porównać te wartości z dokładnymi wartościami pochodnej.

7.2 Rozwiązanie

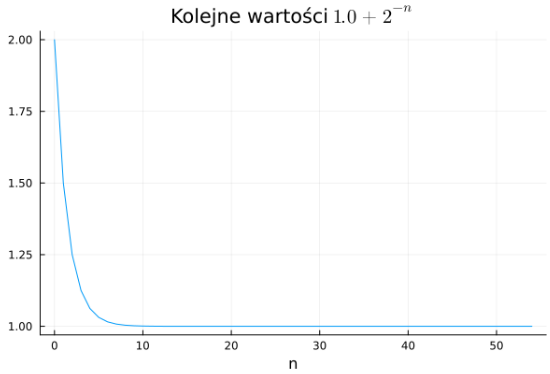
Pochodną funkcji f jest funkcja $f'(x) = \cos x - 3\sin 3x$.
Zaimplementowano procedury obliczające pochodną w punkcie korzystając ze wzoru przybliżonego oraz dokładnego. Porównano otrzymane wyniki oraz przeanalizowano błędy bezwzględne dla n od 0 do 54. Zbadano także zachowanie wartości $1 + h$.

7.3 Wyniki i wnioski

Wyniki zostały przedstawione na poniższych wykresach:

Rysunek 1: Obliczone wartości dokładne i przybliżone

Rysunek 2: Błąd pomiędzy wartością dokładną a obliczoną



Rysunek 3: Badanie wartości $1.0 + h$

Początkowe zmiejszanie wartości h poprawia jakość przybliżenia. Jednak gdy n jest przewyższa 40 błąd rośnie. Powodem takiego zachowania może być fakt, że dodawanie małej wartości do 1.0 nie zwiększa znacząco jej wartości przez co odejmowane są liczby bliskie siebie c generuje duży błąd. Dodatkowo dzielenie przez bardzo małe wartości powiększa błąd.