20.10.2019

Sprawozdanie do pierwszej listy z Obliczeń Naukowych

Tomasz Beneś

Zadanie 1.

a) Opis problemu

Używanie skończonej ilości znaków do zapisu liczby rzeczywistej wpływa na jego dokładność. Szeroko wykorzystywany standard zmiennopozycyjny IEEE-754 również posiada swoje ograniczenia. Celem jest ich znalezienie.

b) Rozwiązanie

Jedną z najważniejszych własności typów zmiennopozycyjnych jest epsilon maszynowy – najmniejsza liczba, która dodana do jedynki w danej arytmetyce da liczbę większą. Żeby znaleźć ε, sprawdzamy coraz mniejsze liczby, aż któraś będzie zbyt mała, żeby zwiększyć wartość jedynki. Dzielenie od jedynki przez dwa ma największy sens ze względu na binarną reprezentację liczb.

Kolejną rozpatrywaną liczbą, nazwaną w zadaniu η, jest najmniejsza liczba po zerze. Znając ją, można wyznaczyć granice przedziału, w którym liczby są na tyle małe, by zostały uznane za zero. Strategia jest podobna do tej przy liczbie ε.

Arytmetyka zmiennopozycyjna zakłada normalizowanie zapisu liczby, co sprawia, że η nie jest uznawana za najmniejszą liczbę większą od zera w standardzie IEEE-754. Liczbą tą będzie taka, której mantysa zaczyna się od jedynki.

Ostatnią rozpatrywaną wartością jest maksymalna liczba, jaką można zapisać w danym typie. Ponieważ zapis jest binarny, strategia polega na znalezieniu pierwszej potęgi dwójki, która jest uznana przez standard za nieskończoność – a następnie zsumowanie wszystkich poprzednich potęg, co da liczbę dokładnie o jeden mniejszą niż umowna nieskończoność.

c) Wyniki

Typ	Macheps	Eta	MIN_{nor}	MAX
Float16	0.000977	6.0e-8	6.104e-5	6.55e4
Float32	1.1920929e-7	1.0e-45	1.1754944e-38	3.4028235e38
Float64	2.220446049250313e-16	5.0e-324	2.2250738585072014e-308	1.7976931348623157e308

Wszystkie obliczone przez program wartości pokrywają się z tymi zwracanymi przez funkcje języka Julia oraz języka C. W formalnym zapisie wartości zmiennopozycyjnych Macheps jest równy ε, a Eta MIN_{sub}.

d) Wnioski

Wszystkie obliczenia w systemie zmiennopozycyjnym, a w szczególności porównania, powinny uwzględniać fakt, że reprezentacje liczb są właściwie przedziałami o różnej długości, w których liczba jest odpowiednio zaokrąglana.

Zadanie 2.

a) Opis problemu

Sprawdzenie, czy wzór Kahana 3 (4/3 – 1) – 1 na macheps daje prawidłowe wyniki.

b) Rozwiązanie

Zaimplementowanie wzoru w języku Julia, ze szczególnym uwzględnieniem kolejności wykonywanych działań.

c) Wyniki

Wzór Kahana daje wyniki zgodne z oczekiwaniami co do modułu.

Typ	Macheps (Kahan)	Macheps (funkcja eps)
Float16	-0.000977	0.000977
Float32	1.1920929e-7	1.1920929e-7
Float64	-2.220446049250313e-16	2.220446049250313e-16

Niedokładność znaku wynika ze sposobu zaokrągleń w standardzie IEEE-754. Konkretnie, przy dzieleniu 4/3 pojawia się wynik bez skończonego zapisu binarnego, z rozwinięciem (01) i ilość cyfr przeznaczonych na mantysę (jej parzystość) zadecyduje o zaokrągleniu w dół lub w górę ostatniego bitu. Samo zaokrąglenie sprawia jednak, że wyrażenie matematyczne równe zeru nie będzie takie w arytmetyce fl.

d) Wnioski

Wzór Kahana wykorzystuje niedokładność zapisu liczby 4/3 w celu policzenia macheps.

Zadanie 3.

a) Opis problemu

Celem jest sprawdzenie, czy liczby w arytmetyce Float64 w odpowiednich przedziałach o granicach będących kolejnymi potęgami dwójki są rozłożone równomiernie.

b) Rozwiązanie

Analiza zapisu binarnego różnych liczb z danego przedziału.

c) Wyniki

Liczba 1 w Float64 ma zapis binarny:

```
0 01111111111 0000000000000000000000000000000000000000000000000000
```

Wykładnik (po wyrównaniu bias) jest równy zeru, a mantysa jedynce (w zapisie tylko część ułamkowa). Kolejna liczba w Float64 to:

```
0 01111111111 0000000000000000000000000000000000000000000000000001
```

Natomiast ostatnia liczba przed dwójką ma zapis:

```
0 01111111111 1111111111111111111111111111111111111111111111111111
```

Można zauważyć, że zapełniona została mantysa, mająca 52 znaki, co daje 2^{52} dodawań jedynki, by ją zapełnić. W następnym kroku zwiększa się wykładnik.

```
0 10000000000 0000000000000000000000000000000000000000000000000000
```

W przypadku innych przedziałów, jak [0,5; 1,0] można zaobserwować podobne zachowanie:

```
0 01111111110 0000000000000000000000000000000000000000000000000000
0 01111111110 0000000000000000000000000000000000000000000000000001
0 01111111110 1111111111111111111111111111111111111111111111111111
0 01111111111 0000000000000000000000000000000000000000000000000000
```

Ale tutaj wykładnik to –1, dlatego krok δ to nie 2^{-52} a $2^{-52} \cdot 2^{-1} = 2^{-53}$.

Dla przedziału [2,0; 4,0] otrzymujemy:

```
0 10000000000 0000000000000000000000000000000000000000000000000000
0 10000000000 0000000000000000000000000000000000000000000000000001
0 10000000000 1111111111111111111111111111111111111111111111111111
0 10000000001 0000000000000000000000000000000000000000000000000000
```

Wykładnik to 2, stąd krok $\delta = 2^{-52} \cdot 2^{1} = 2^{-51}$.

d) Wnioski

Im bliżej zera, tym mniejsze są odstępy między kolejnymi reprezentowanymi liczbami. W ogólności, dla typu Float64, krok δ dla przedziału w postaci $[2^a; 2^{a+1}]$ wyraża się wzorem:

$$\delta = 2^{-52} \cdot 2^{-a}$$

Zadanie 4.

a) Opis problemu

Ze względu na niedokładność zapisu w arytmetyce fl, niektóre matematycznie poprawne i trywialne równości stają się nieprawdziwe. Jedną z nich jest:

$$x \cdot (1/x) = 1$$

b) Rozwiązanie

Znalezienie liczny x, która nie spełnia powyższego równania, polega na sprawdzaniu w pętli kolejnych wartości Float64 począwszy od jedynki. Pierwsza napotkana liczba, która nie spełnia równości, będzie jednocześnie najmniejszą taką liczbą w zadanym przedziale.

c) Wyniki

Znaleziona liczba to 1.000000057228997.

d) Wnioski

Sprawdzając równości należy uwzględniać błąd wynikający z reprezentacji i działań arytmetyki. Często praktyką są epsilon-równości, dla których można ustawić czułość.

Zadanie 5.

a) Opis problemu

Przejrzenie algorytmów obliczania iloczynu skalarnego i ocena ich skuteczności.

b) Rozwiązanie

Podane algorytmy zostały zaimplementowane w języku Julia i wykonane dla Float32 i Float64.

c) Wyniki

Algorytm	Float32	Float64
a	-0.4999443	1.0251881368296672e-10
b	-0.4543457	-1.5643308870494366e-10
c	5.5190055e6	5.519005839180414e6
d	-0.5	0.0

Błąd względny:

Algorytm	Float32	Float64
a	4.966805771498**e10**	-1.11849553139816**e1**
b	4.513796526955**e10**	1.45411866451659**e1**
c	-5.48**e17**	-5.4829768147225**e17**
d	4.967359135306**e10**	-1**e0**

Wyjątkowo złe wyniki (błąd rzędu 10^{17}) dał algorytm **c** polegający na sumowaniu od największego do najmniejszego. Tak duża różnica spowodowana jest „pochłanianiem" małych wartości przy dodawaniu ich do dużych, spowodowanych wyrównywaniem wykładników. Podwójna precyzja pozwoliła pozbyć się w przypadku najlepszego algorytmu **d** większej części błędu, niestety algorytm uznał wektory za równoległe, co mogłoby prowadzić do niebezpieczeństw.

d) Wnioski

Zadanie liczenia iloczynu skalarnego jest źle uwarunkowane. Podanie dwóch prawie równoległych wektorów będzie dawać fałszywe wyniki ze względu na zbyt małą precyzję obliczeń.

Ponadto, można zaobserwować, że dodawanie liczb od najmniejszej do największej powoduje najmniejszą kumulację błędów.

Zadanie 6.

a) Opis problemu

Policzenie wartości tej samej funkcji za pomocą innych algorytmów.

b) Rozwiązanie

Podane algorytmy zostały zaimplementowane w języku Julia i wykonane dla Float64.

c) Wyniki

Dokładne wyniki znajdują się w pliku z6r.ods.

Począwszy od 8^{-9} algorytm funkcji f zwraca 0.0, co jest niepoprawnym wynikiem. Dzieje się tak z powodu przekłamania, które powstaje przy odejmowaniu liczb bliskich sobie. Błąd względny odejmowania tych liczb wynika z wcześniejszego zaokrąglenia wartości pierwiastka. Równoważny matematycznie wzór g omija ten problem i jest lepszy, bo choć wykonuje więcej działań, nie są one obarczone takim błędem jak odejmowanie bliskich liczb.

d) Wnioski

Należy unikać odejmowania bliskich sobie liczb. Można to osiągać stosując równoważne matematycznie formuły. Ponieważ błąd wynika z zaokrąglenia, wydłużenie mantysy również mogłoby pomóc.

Zadanie 7.

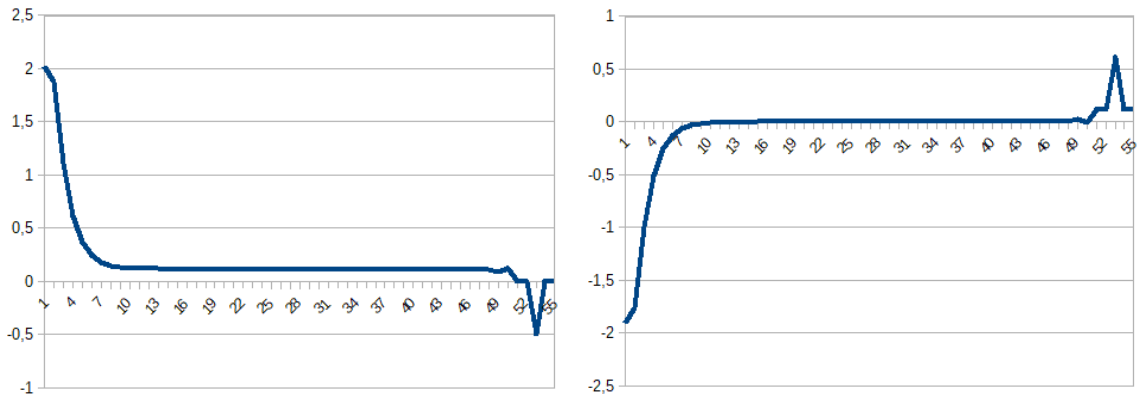
a) Opis problemu

Obliczanie przybliżonej wartości pochodnej funkcji w punkcie.

b) Rozwiązanie

Zastosowanie wzoru podanego w zadaniu.

c) Wyniki



Po lewej – wykres wartości f' dla kolejnych $h = 2^{-n}$ (n = 0, 1, …, 54).
Po prawej – błędy kolejnych przybliżeń.

W pewnym momencie zmniejszanie h przestaje poprawiać przybliżenie, bo błąd zaczyna wynikać z niedokładności odejmowania f(x) i f(x+h) – liczb bliskich sobie. Na końcu można zaobserwować natomiast zawahanie wynikające z utraty części ułamkowej x+h (w pewnym momencie x+h=x).

d) Wnioski

Dla niektórych funkcji jest to dobry sposób aproksymacji jej pochodnej. Problem może pojawić się w przypadku funkcji prawie stałych, gdzie $f(x) \approx f(x+h)$ i odejmowanie tych liczb spowoduje nagromadzenie sporego błędu, lub funkcja zostanie uznana za stałą kiedy nie powinna.

Z tego samego powodu należy uważać z wielkością h – zbyt małe, zwiększa ryzyko popełnienia wspomnianych błędów.