Obliczenia Naukowe

Sprawozdanie z laboratorium nr 1

Piotr Kawa

Wrocław 11.10.2017

0. Wprowadzenie

Celem listy zadań było zapoznanie się z podstawowymi zagadnieniami dotyczącymi obliczeń naukowych. Na końcu każdego podrozdziału znajdują się wyniki otrzymane dzięki obliczeniom programów napisanych w języku Julia.

1. Zadanie pierwsze

1.1 Opis zadania

Celem zadania było iteracyjne wyliczenie kilku wartości: epsilonu maszynowego (zwanego również $macheps$), eta (najmniejsza dodatnia liczba rzeczywista) oraz liczbę MAX dla trzech typów liczb zmiennopozycyjnych – $Float16$ (half), $Float32$ (single), $Float64$ (double). Otrzymane wartości musiały być następnie porównane ze stałymi ze środowiska Julia.

1.2 Rozwiązanie

1.2.1 Epsilon maszynowy

W celu wyliczenia epsilonu maszynowego zastosowano operację dzielenia w pętli przez 2. Liczbą startową było 1.0. Operacja była wykonywana tak długo dopóki nie został spełniony następujący warunek:

$$1.0 + \frac{macheps}{2} = 1.0$$

1.2.2 eta

Strategia dotycząca wyliczenia eta, czyli najmniejszej dodatniej liczby rzeczywistej, była podobna. Zastosowane zostało dzielenie w pętli przez dwa. Tym razem natomiast warunkiem końcowym było:

$$\frac{eta}{2} \leq 0.0$$

1.2.3 MAX

Wyliczenie maksymalnej liczby z każdego zakresu wymagało zastosowania innej strategii. W przypadku Float16 i Float32 do liczby 1.0 dodawana była najmniejszą wartość, która w danym momencie powoduje spełnienie warunku $current + next > current$. Z początku jest to $macheps$, który następnie ze względu na pochłanianie musi być mnożony co pewien czas przez 2. Operacja była powtarzana dopóki dopóty wynik nie był nieskończonością.

W przypadku Float64 – samo mnożenie liczby przez 2 dawało błędne wyniki, poprzednie podejście natomiast trwało o wiele za długo. Rozwiązaniem było wymnożenie liczby 1.0 przez 2 dopóki dopóty znajdować się będziemy w zakresie $Float64$. Następnie do aktualnego wyniku dodawana była jak największa liczba, która nie powodowała wyjścia poza zakres. Polegało to na sprawdzeniu czy liczba dodana do wyniku nie jest jeszcze nieskończonością, jeśli była – liczba nie była dodawana, a operacja była wykonywana na liczbie 2 razy mniejszej, jeśli mieściła się w zakresie – dodawana była do wyniku.

1.3 Wyniki

1.3.1 $macheps$

Typ	$computeEpsilon()$	$eps()$
$Float16$	$9.765625e-4$	$9.765625e-4$
$Float32$	$1.1920929e-7$	$1.1920929e-7$
$Float64$	$2.22044604\mathrm{e}-16$	$2.22044604\mathrm{e}-16$

1.3.2 eta

Typ	$computeETA()$	$nextfloat(0.0)$
$Float16$	$5.960464e-8$	$5.960464e-8$
$Float32$	$1.4012984e-45$	$1.4012984e-45$
$Float64$	$4.9\mathrm{e}-324$	$4.9\mathrm{e}-324$

1.3.3 MAX

Typ	$computeMax()$	$realmax()$
$Float16$	$6.5504e4$	$6.5504e4$
$Float32$	$3.4028235\mathrm{e}38$	$3.4028235\mathrm{e}38$
$Float64$	$1.79769313e308$	$1.79769313e308$

Jak wynika z powyższych tabel obliczone wartości zgadzają się z wynikami funkcji $eps()$ dla wartości $macheps$, $nextfloat(0.0)$ dla eta oraz $realmax()$ dla MAX.

Jednym z poleceń było również porównanie wartości $macheps$ i eta z tymi, które można znaleźć w pliku nagłówkowym $float.h$ dowolnej instalacji języka C. Poniższa tabela pokazuje, że mają one te same wartości dla $Float32$ oraz $Float64$. C nie wspiera $Float16$.

Wartość	$single\ (Float32)$	$double\ (Float64)$
$macheps$	$1.1920929e-07$	$2.22044604e-16$
MAX	$3.4028235e38$	$1.79769313e308$

1.4 Wnioski

Jednym z zadanych pytań było podanie związku liczby $macheps$ z precyzją arytmetyki oznaczaną literą ε. Jak wiemy precyzja arytmetyki to maksymalny błąd względny podczas zaokrąglenia liczby rzeczywistej do liczby zmiennoprzecinkowej. Tym samym jest również epsilon maszynowy.

Kolejnym pytaniem był związek liczby eta z liczbą MIN_{sub}. Okazuje się, że ponownie związek jest następujący – oba pojęcia są równoważne. Potwierdzenie tego stwierdzenia możemy znaleźć sięgając do raportu Williama Kahana na temat standardu IEEE 754: *„Subnormals, [..], are nonzero numbers with an unnormalized significand n and the same minimal exponent k as is used for 0”*.

2. Zadanie drugie

2.1 Opis zadania

Celem zadania drugiego było stwierdzenie za pomocą eksperymentu czy William Kahan miał rację twierdząc, że wyrażenie $3\left(\frac{4}{3}-1\right)-1$ pozwala na obliczenie epsilonu maszynowego w arytmetyce zmiennopozycyjnej.

2.2 Wyniki

Wyrażenie obliczone zostało dla każdej z trzech arytmetyk. Wyniki były następujące:

	$\boldsymbol{Float16}$	$\boldsymbol{Float32}$	$\boldsymbol{Float64}$
$3\left(\frac{4}{3}-1\right)-1$	-0.000977	$1.1920929e-7$	$-2.220446049250313e-16$

2.3 Wnioski

Wynikiem działania $3\left(\frac{4}{3}-1\right)-1$ jest liczba zero. Komputer jednak nie jest w stanie przedstawić tego rozwiązania poprawnie – w działaniu występuje liczba $\frac{4}{3}$, której rozwinięcie dziesiętne jest nieskończone. Maszyna musi zaokrąglić tą liczbę do najbliższej liczby zmiennopozycyjnej w arytmetyce, w której aktualnie operuje co powoduje powstanie błędu. Wartość $macheps$ jest maksymalnym błędem względnym, który może zajść podczas zaokrąglania liczb. Powyższe wnioski są niezbitym dowodem na to, że stwierdzenie Kahana jest poprawne tylko dla $Float32$.

3. Zadanie trzecie

3.1 Opis zadania

Celem tego zadania było eksperymentalne sprawdzenie rozmieszczenia liczb zmiennopozycyjnych Float64 w przedziałach $[1, 2]$, $\left[\frac{1}{2}, 1\right]$ oraz $[2, 4]$.

3.2 Rozwiązanie i wyniki

Do rozwiązania tego zadania nieocenioną pomocą była funkcja $bits()$, która zwraca binarną reprezentację danej liczby. Eksperyment polegał na kilkurazowym zwiększeniu najmniejszej liczby w danych przedziałach o $\delta = 2^{-52}$.

Wartość	$bits()$
1.0	0011111111110000000000000000000000000000000000000000000000000000
$1.0 + 2^{-52}$	0011111111110000000000000000000000000000000000000000000000000001
$0.5 + 2 * 2^{-52}$	0011111111110000000000000000000000000000000000000000000000000010
0.5	0011111111100000000000000000000000000000000000000000000000000000
$0.5 + 2^{-52}$	0011111111100000000000000000000000000000000000000000000000000010
$0.5 + 2 * 2^{-52}$	0011111111100000000000000000000000000000000000000000000000000100
2.0	0100000000000000000000000000000000000000000000000000000000000000
$2.0 + 2^{-52}$	0100000000000000000000000000000000000000000000000000000000000000
$2.0 + 2 * 2^{-52}$	0100000000000000000000000000000000000000000000000000000000000001

3.3 Wnioski

Jak widać po powyższych wynikach liczby zmiennopozycyjne w podanych przedziałach są oddalone od siebie o pewien krok. Zwiększenie liczby 1.0 o 2^{-52} spowodowało zwiększenie się jej o 1 bit, w przypadku 0.5 o 2, natomiast dodanie δ do 2.0 nie zaowocowało żadną zmianą (dopiero kolejne dodanie zwiększyło o 1 bit).

Powód tego stanu rzeczy jest wyjaśniony w książce Kincaida, Cheneya „Analiza Numeryczna”: *„Rozkład liczb zmiennopozycyjnych w komputerze jest nierównomierny. Między kolejnymi potęgami dwójki znajduje się tyle samo liczb maszynowych. Dlatego znaczna ich część skupia się w pobliżu zera*”.

4. Zadanie czwarte

4.1 Opis zadania

Celem było znalezienie w arytmetyce $Float64$ takiej liczby zmiennopozycyjnej x, która w przedziale $1 < x < 2$ spełnia warunek $x * \frac{1}{x} \neq 1$. Kolejny podpunkt wymagał znalezienia najmniejszej takiej liczby.

4.2 Rozwiązanie

W celu znalezienia liczby spełniającej powyższą nierówność użyta została operacja, która wykonywana była w pętli od 1.0 do 2.0. Za każdym przejściem zwiększała ona liczbę (startowo 1.0) o epsilon maszynowy. Gdy ta spełniała warunek $x * \frac{1}{x} \neq 1$ wypisywana była na ekranie.

4.3 Wyniki

Najmniejszym ze zwróconych wyników była liczba 1.000000057228997.

4.4 Wnioski

Istnienie takich liczb jak powyższa wynika z niemożności przedstawienia przez komputer każdej liczby rzeczywistej w postaci $\frac{1}{x}$. W takiej sytuacji jest ona zaokrąglana do najbliższej liczby zmiennopozycyjnej. Nie jest to jednostkowy przypadek, gdyż istnieje wiele takich liczb, o czym świadczy mnogość wyników zwróconych przez opisywaną wcześniej funkcję.

5. Zadanie piąte

5.1 Opis zadania

Celem zadania piątego było obliczenie iloczynu skalarnego dwóch wektorów. Należało to zrobić za pomocą 4 algorytmów używając precyzji $Float32$ oraz $Float64$, a następnie porównać z prawidłowym wynikiem tejże operacji, czyli $1.00657107000000e - 11$.

5.2 Rozwiązanie

Pierwszy algorytm polegał na wyliczeniu iloczynu metodą „w przód”: $S = S + x_i * y_i$ w pętli $1..5$, gdzie x i y to wektory. Druga metoda („w tył”) prowadziła pętlę od 5 do 1. Trzeci algorytm polegał na dodaniu do siebie dodatnich wyników mnożenia $x_i * y_i$ w kolejności od największego do najmniejszego, ujemnych natomiast odwrotnie. Następnie należało obliczyć sumę obydwu z nich. Ostatnia metoda była odwrotnością trzeciej.

5.3 Wyniki

Algorytm	$Float32$	$Float64$
1.	-0.499944	$1.0251881368e - 10$
2.	-0.454345	$-1.5643308870e - 10$
3.	-0.5	0.0
4.	-0.5	0.0

5.4 Wnioski

Żaden z przedstawionych algorytmów nie dał poprawnego wyniku. W przypadku dwóch pierwszych metod wyniki przy użyciu liczb pojedynczej precyzji ($Float32$) obarczone były dużym rozmiarem błędu. Sytuacja wyglądała lepiej przy podwójnej precyzji – jednakowoż nadal są to wartości odległe od wartości rzeczywistej. Sytuacja w przypadku algorytmu trzeciego i czwartego wyglądała jeszcze gorzej – nawet w reprezentacji $Float64$.

Najważniejszym wnioskiem płynącym z tego ćwiczenia jest to, że kolejność operacji wykonywanych w komputerze ma znaczenie co przekłada się na to, że nie każdy algorytm mimo, iż z logicznego puntu widzenia jest poprawny, zwróci nam dobre wyniki dla każdych danych.

6. Zadanie szóste

6.1 Opis zadania

Celem tego zadania było obliczenie w arytmetyce Float64 wartości dwóch funkcji dla $x = 8^{-1}, 8^{-2}$ itd. Funkcje te wyglądały następująco:

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

6.2 Rozwiązanie i wyniki

W pętli wyliczane były wartości funkcji $f(x)$ oraz $g(x)$ dla kolejnych wartości 8^k.

x	$f(x)$	$g(x)$	Wolfram Alpha
8^0	$4.14213562373095145e-1$	$4.142135623730950899e-1$	$4.142135623730950\ e-1$
8^{-1}	$7.78221853731864143e-3$	$7.782218537318706490e-3$	$7.7822185373187065e-3$
8^{-2}	$1.2206286282867573e-4$	$1.220628628287590136e-4$	$1.2206286282875902e-4$
8^{-3}	$1.9073468138230964e-6$	$1.907346813826565901e-6$	$1.9073468138265658e-6$
8^{-4}	$2.9802321943606102e-8$	$2.98023219436061159e-8$	$2.9802321943606115e-8$
8^{-5}	$4.65661287307739e-10$	$4.656612871993190e-10$	$4.6566128719931904e-10$
8^{-6}	$7.2759576141834e-12$	$7.2759576141570e-12$	$7.2759576141569561e-12$
8^{-7}	$1.136868377216e-13$	$1.136868377216e-13$	$1.1368683772160956e-13$
8^{-8}	$1.7763568394e-15$	$1.7763567865e-15$	$1.7763568394002488e-15$
8^{-9}	0.0	$2.77555755e-17$	$2.7755575615628913e-17$
8^{-10}	0.0	$4.336809e-19$	$4.3368086899420177e-19$
8^{-11}	0.0	$6.7763e-21$	$6.77626357803440271e-21$

6.4 Wnioski

Podczas analizy danych zwróconych przez funkcje $f(x)$ oraz $g(x)$ porównane zostały one z wynikami otrzymanymi dzięki użyciu pakietu matematycznego Wolfram Alpha. Wyniki te również zostały zamieszone w powyższej tabeli.

Z matematycznego punktu widzenia funkcje $f(x)$ oraz $g(x)$ powinny zwracać te same wyniki. Tabela z wynikami pokazuje natomiast, że tak nie jest. Przyczyną takiego stanu rzeczy jest odejmowanie liczby 1, które znajduje się w funkcji $f(x)$. Odejmowanie od siebie bardzo podobnych liczb ma negatywny wpływ na wynik tejże operacji – z tego właśnie powodu od pewnego momentu funkcja $f(x)$ zwraca dalekie od prawdziwych wyniki.

7. Zadanie siódme

7.1 Opis zadania

Zadanie siódme polegało na obliczeniu wartości pochodnej funkcji $f(x) = \sin x + \cos 3x$ w punkcie $x_0 = 1$. Należało użyć wzoru $f'(x) = \frac{f(x_0+h)-f(x_0)}{h}$, gdzie $h = 2^{-n}$ $(n = 0,1,\ldots,54)$. Ponadto należało obliczyć wartość błędu $\left|f'(x_0) - \widetilde{f}'(x_0)\right|$. Arytmetyką obliczeń był $Float64$.

7.2 Rozwiązanie

W celu obliczenia wartości pochodnej stworzona została funkcja $computeApproximateDerivative()$, która wyliczała wartość $\widetilde{f}'(x_0)$. W pętli przekazywane były jej wartości h, czyli delta - obliczenia te były wykonane 55 razy (dla $n = 0..54$). Funkcja $computeError(x, y)$ wyliczała różnicę przekazanych jej argumentów – $f'(x_0)$ oraz $\widetilde{f}'(x_0)$.

7.3 Wyniki

$f'(x) = \cos(x) - 3\sin(3x)$, a więc $f'(1) \approx 0.116942281$

h	Wynik funkcji $computeApproximateDerivative()$	Wartość błędu $\left\|f'(x_0) - \widetilde{f}'(x_0)\right\|$
2^{0}	2.0179892252685967	1.9010469435800585
2^{-1}	1.8704413979316472	1.753499116243109
2^{-2}	1.1077870952342974	$9.908448135457593\mathrm{e}-1$
2^{-3}	$6.232412792975817\mathrm{e}-1$	$5.062989976090435\mathrm{e}-1$
2^{-4}	$3.704000662035192\mathrm{e}-1$	$2.534577845149810\mathrm{e}-1$
2^{-16}	$1.170038392883725\mathrm{e}-1$	$6.155759983439423\mathrm{e}-5$
2^{-17}	$1.169730604597134\mathrm{e}-1$	$3.077877117529936\mathrm{e}-5$
2^{-35}	$1.169395446777343\mathrm{e}-1$	$2.737010803777195\mathrm{e}-6$
2^{-36}	$1.169433593750000\mathrm{e}-1$	$1.077686461847804\mathrm{e}-6$
2^{-37}	$1.169281005859375\mathrm{e}-1$	$1.418110260065219\mathrm{e}-5$
2^{-49}	0.125	$8.057718311461848\mathrm{e}-3$
2^{-50}	0.0	$1.169422816885381\mathrm{e}-1$
2^{-51}	0.0	$1.169422816885381\mathrm{e}-1$
2^{-52}	-0.5	$6.169422816885382\mathrm{e}-1$
2^{-53}	0.0	$1.169422816885381\mathrm{e}-1$
2^{-54}	0.0	$1.169422816885381\mathrm{e}-1$

7.4 Wnioski

Jak widać zmniejszanie wartości h przestaje od pewnego momentu (pomiędzy 2^{-35}, a 2^{-37}) przybliżać wynik, a wręcz przeciwnie – zaczyna generować coraz większy błąd. Taki stan rzeczy jest wynikiem operacji na bliskich sobie liczbach w arytmetyce zmiennopozycyjnej, które generują spore błędy. Od momentu, gdy $h = 2^{-50}$ otrzymany błąd nie zmienił się, co było wynikiem przekroczenia przez h liczby $macheps$.