Obliczenia Naukowe

Prowadzący: dr hab. Paweł Zieliński, prof. PWr

Lista 1

Autor: Daniel Drapała 244939

Przybliżenie zagadnienia arytmetyki zmiennopozycyjnej (IEEE 754), częściowe zrozumienie, a także pułapki na które możemy trafić podczas korzystania z niej.

Zadanie 1 Epsilon Maszynowy

Epsilonem maszynowym macheps (ang. machine epsilon) nazywamy najmniejszą liczbę macheps > 0 taką, że 1.0 $\oplus$ macheps > 1.0.

Polecenie

Iteracyjne wyznaczenie epsilonów maszynowych dla wszystkich typów zmiennopozycyjnych i porównanie ich z wartościami zwracanymi przez funkcję eps() z języka Julia oraz z danymi zawartymi w pliku nagłówkowym float.h języka C.

Rozwiązanie

Prosta pętla w której dzielę zmienną meps (początkowo równą 1) przez dwa aż do momentu w którym $1 + meps \div 2 = 0$.

Wyniki

Tabela 1 przedstawia wyniki poszczególnych sposobów pokazania macheps.

Typ	macheps wyliczony	eps(typ)	float.h
Float16	0.0009765625	0.0009765625	Brak
Float32	1.192092895507813 10^{-7}	1.192092895507813 10^{-7}	$1.1920929 \cdot 10^{-7}$
Float64	$2.220446049250313 \cdot 10^{-16}$	$2.220446049250313 \cdot 10^{-16}$	$2.220446049250313 \cdot 10^{-16}$

Tabela 1: Zestawienie wyliczonych epsilonów maszynowych

Wnioski

Wyniki macheps wyliczonego iteracyjnie za pomocą prostej pętli pokrywają się z danymi zawartymi w pliku float.h i funkcją eps(). Epsilon maszynowy wiąże się z precyzją arytmetyki (zadaną wzorem 2^{-t-1}), gdzie $t - długość\ mantysy$, jego wartość jest wtedy dwa razy większa (2^{-t}).

Liczba Eta

Polecenie

Napisać program w języku Julia wyznaczający iteracyjnie liczbę eta taką, że eta > 0.0 dla wszystkich typów zmiennopozycyjnych.

Rozwiązanie

Prosta pętla, która kończy dzielenie jedynki przez dwa w momencie w którym otrzymamy zero i zwraca wynik ostatniego dzielenia przez 2.

Wyniki

	eta	nextfloat()	MIN_{nor} /floatmin()
Float16	$5.96046447753906310^{-08}$	$5.96046447753906310^{-08}$	$6.104\ 10^{-5}$
Float32	$1.401298464324817\ 10^{-45}$	$1.401298464324817\ 10^{-45}$	$1.1754944\ 10^{-38}$
Float64	$4.940656458412465\ 10^{-324}$	$4.940656458412465\ 10^{-324}$	$2.2250738585072014\ 10^{-308}$

Wnioski

Po wykonaniu funkcji bitstring na otrzymanym wyniku możemy zauważyć że jest to najmniejsza liczba możliwa do zapisania w IEEE 754.
0000000000000000000000000000000000000000000000000000000000000001
Dodatkowo wszystkie bity cechy są wyzerowane ponieważ jest to liczba zdenormalizowana (stąd nazwa MIN_{sub} od min subnormal).
Za to MIN_{nor} jest minimalną znormalizowaną liczbą, czyli cecha nie jest wyzerowana:
0000000000010000000000000000000000000000000000000000000000000000

Pytania i Odpowiedzi

1. Jaki związek ma liczba eta z liczbą MIN_{sub}?
2. Co zwracają funkcje floatmin(Float32) i floatmin(Float64) i jaki jest związek zwracanych wartości z liczbą MIN_{nor}?

Ad.1 Liczba eta jest najmniejszą możliwą liczbą po zerze, więc razem z MIN_{sub} Tworzą granicę tzw. „zera maszynowego” $(-MIN_{sub}, MIN_{sub})$

Ad.2 Funkcje te zwracają najmniejszą wartość nieujemną postaci normalnej (MIN_{nor} jest większe od MIN_{sub}) to znaczy że jest to najmniejsza liczba do zapisania i będąca dalej liczbą normalną

Liczba Max

Polecenie

Napisać program w języku Julia wyznaczający iteracyjnie liczbę maksymalną możliwą do zapisania w danej arytmetyce dla 3 typów i porównanie jej z wartościami zwracanymi prez floatmax() i przechowywanymi w pliku float.h

Rozwiązanie

Prosta pętla, która mnoży zmienną 1.0 × 2.0 aż do momentu otrzymania nieskończoności. Wtedy wynikiem ostateczny będzie pomnożenie liczby z ostatniej iteracji razy (2.0-macheps())

Wyniki

	Max	floatmax()	$float.h$
Float16	6.550400000000000 10^{04}	6.550400000000000 10^{04}	–
Float32	3.402823466385289 10^{38}	3.402823466385289 10^{38}	3.402823466385289 10^{38}
Float64	1.797693134862316 10^{308}	1.797693134862316 10^{308}	1.797693134862316 10^{308}

Tabela 2 wyniki liczby MAX

Wnioski

Wyniki pokrywają się, a więc metoda znajdywania max jest poprawna.

Zadanie 2 Epsilon maszynowy wg. Kahana

Kahan stwierdził że macheps można uzyskać za pomocą obliczenia wyrażenia $3\left(\frac{4}{3}-1\right)-1$ w arytmetyce zmiennopozycyjnej.

Polecenie

Eksperymentalnie obliczyć wyrażenie Kahana i porównać z wynikiem funkcji eps().

Porównanie wyników z eps(typ):

	Macheps by Kahan	Eps(t)
Float16	-9.765625000000000000 10^{-4}	9.765625000000000000 10^{-04}
Float32	1.19209289550781250000 10^{-07}	1.19209289550781250000 10^{-07}
Float64	-2.22044604925031308085 10^{-16}	2.22044604925031308085 10^{-16}

Tabela 3 Wyniki obliczonego machepsa sposobem Kahana

Wnioski:

Możemy zauważyć, że wartości pokrywają się, zmieniony jest jedynie znak w typach Float16 i Float64. Wiązane jest to z zasadą *„round to even”*, która to w zależności od parzystości mantysy zaokrągla liczbę z niedomiarem lub z nadmiarem (zero na ostatniej pozycji mantysy – zaokrąglenie z niedomiarem, jeden – z nadmiarem). Więc we Float16 i Float64 dostajemy zaokrąglenie z niedomiarem, a w Float32 z nadmiarem. A to dlatego że całe to równanie w reprezentacji dwójkowej jest liczbą okresową (nie istnieje skończona liczba bitów określająca daną liczbę).

Zadanie 3

Polecenie

W zadaniu należy eksperymentalnie sprawdzić w języku Julia, że w arytmetyce Float64 liczby zmiennopozycyjne są równomiernie rozmieszczone w przedziale $[1, 2]$, a także ich rozmieszczenie w przedziałach $[\frac{1}{2}, 1]$ i $[2, 4]$.Opis zadania

Rozwiązanie

Prosta pętla for zaczynająca się od lewej strony przedziału, dodająca do przedziału liczbę 2^{-52}. Wnioski budowane będą na podstawie reprezentacji bitowej poszczególnych iteracji.

Wyniki

Pierwsze cztery iteracje na granicach przedziału dla [1,2] i $\delta=2^{-52}$
0011111111110000000000000000000000000000000000000000000000000001
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000011
0011111111110000000000000000000000000000000000000000000000000100
…
0011111111111111111111111111111111111111111111111111111111111100
0011111111111111111111111111111111111111111111111111111111111101
0011111111111111111111111111111111111111111111111111111111111110
0011111111111111111111111111111111111111111111111111111111111111
Pierwsze cztery iteracje na granicach przedziału dla $[\frac{1}{2},1]$ i $\delta=2^{-53}$
0011111111100000000000000000000000000000000000000000000000000001
0011111111100000000000000000000000000000000000000000000000000010
0011111111100000000000000000000000000000000000000000000000000011
0011111111100000000000000000000000000000000000000000000000000100
…
0011111111101111111111111111111111111111111111111111111111111100
0011111111101111111111111111111111111111111111111111111111111101
0011111111101111111111111111111111111111111111111111111111111110
0011111111101111111111111111111111111111111111111111111111111111
Pierwsze cztery iteracje na granicach przedziału dla [2,4] i $\delta=2^{-51}$
0011111111100000000000000000000000000000000000000000000000000001
0011111111100000000000000000000000000000000000000000000000000010
0011111111100000000000000000000000000000000000000000000000000011
0011111111100000000000000000000000000000000000000000000000000100
…
0011111111101111111111111111111111111111111111111111111111111111
0011111111101111111111111111111111111111111111111111111111111110
0011111111101111111111111111111111111111111111111111111111111101
0011111111101111111111111111111111111111111111111111111111111100

Tabela 4iteracje kolejnych Floatów w 3 różnych przedziałach

Wnioski

Te 3 przedziały nie są przypadkowe, są to przedziały między kolejnymi potęgami dwójki, które dzięki tabelkom wynikowym pozwalają zauważyć, że im większy przedział (im wyższe potęgi liczby 2) tym gęstość liczb maleje. W sprawdzanym bowiem przedziałach nie zmienia się cecha, tylko mantysa. Podsumowując widzimy kolejne iteracje zwiększają pierwszy najmniej znaczący bit o jeden, więc rozmieszczone są równomiernie, lecz im większe wartości, tym mniejsza gęstość ich rozmieszczenia.

Zadanie 4 Nieodwracalność dzielenia

Polecenie

Znalezienie najmniejszej liczby x z przedziału (1,2) w Float64 takiej, że $x \cdot (1 \div x) \neq 1$.

Rozwiązanie

W celu rozwiązania zadania dla kolejnych liczb x w arytmetyce Float64, zaczynając od najmniejszej liczby większej od 1, zostało sprawdzone czy warunek $x \cdot (1 \div x) \neq 1$ zachodzi. W momencie znalezienia pierwszej takiej liczby program przerywał pracę i wyświetlał wynik na ekranie.

Wyniki

Przykładowa liczba znaleziona: 1.5000000000000002 a najmniejsza jest pierwszą napotkaną od jedynki po nextfloat(): 1.000000057228997

Wnioski

Zadanie pokazuje, że działania arytmetyczne na liczbach zmiennopozycyjnych mogą generować błędy związane z zaokrąglaniem wyliczonych wartości. Przy używaniu typów zmiennopozycyjnych takie błędy często są nieuniknione, zwłaszcza przy dzieleniu, które nie jest w tej arytmetyce odwracalne.

Zadanie 5 Obliczanie iloczynu skalarnego

Obliczenie iloczynu skalarnego danych wektorów z wykorzystaniem czterech różnych algorytmów sumowania dla typów Float32 i Float64.

x = [2.718281828, −3.141592654, 1.414213562, 0.5772156649, 0.3010299957]

y = [1486.2497, 878366.9879, −22.37492, 4773714.647, 0.000185049]

Rozwiązanie

W programie zaimplementowano podane algorytmy:

1. ”w przód”: $\sum_{i=1}^{n} x_i y_i$;

2. ”w tył”: $\sum_{i=n}^{1} x_i y_i$;

3. dodanie dodatnich liczb w porządku od największej do najmniejszej oraz ujemnych w porządku od najmniejszej do największej, a następnie dodanie do siebie obliczonych sum częściowych; zostało to wykonane za pomocą sortowania i odpowiedniego dodania elementów tablicy sum częściowych;

4. metoda przeciwna do sposobu 3.

Wyniki

	1	2	3	4
Float32	-0.4999443	-0.4543457	-0,5	-0,5
Float64	$1.0251881368296672\ 10^{-10}$	$-1.5643308870494366\ 10^{-10}$	0.0	0.0

Tabela 5 Iloczyn skalarny wektorów 4 różne algorytmy

Prawidłowy iloczyn skalarny wektorów obliczony bez zaokrąglania danych to $-1.00657107000000 \cdot 10^{-11}$. Wszystkie otrzymane wyniki są od niego różne.

Wnioski

Zadanie pokazuje, że kolejność wykonywania działań nie jest bez znaczenia. Na przykład dodanie do bardzo dużej liczby w stosunku do niej bardzo małej generuje błędy, ponieważ mała liczba zostanie w jakimś stopniu zignorowana podczas zaokrąglania wyniku.

Jednym ze sposobów na uniknięcie dużych błędów, kiedy inne metody zawodzą, jest użycie arytmetyki o większej precyzji. Użycie Float64 zamiast Float32 w zadaniu w znaczący sposób przybliżyło uzyskane wyniki do poprawnego, jednak nawet to nie dało zadowalających rezultatów.

Zadanie 6

Polecenie

Zadanie polega na obliczeniu kolejnych wartości funkcji, które są tożsame, w arytmetyce Float64. Za argumenty wybieramy kolejną ujemną potęgę liczby osiem.

$f(x) = \sqrt{x^2 + 1} - 1$ $g(x) = \frac{x^2}{\sqrt{x^2+1}+1}$ $dla\ x = 8^{-1}, 8^{-2}, 8^{-3}, \ldots$

Wyniki

-Początkowo wartości są do siebie zbliżone

-Funkcja f osiąga wartość zero dla $x=8^{-9}$, kiedy to g osiąga wartość dopiero dla $x=8^{-179}$.

-Pokazane szczegółowo w Tabeli nr 7

8^{-1}	0.0077822185373186414	0.0077822185373187065
8^{-2}	0.00012206286282867573	0.00012206286282875901
8^{-3}	$1.9073468138230965 \cdot 10^{-6}$	$1.907346813826566 \cdot 10^{-6}$
8^{-4}	$2.9802321943606103 \cdot 10^{-8}$	$2.9802321943606116 \cdot 10^{-8}$
8^{-8}	$1.7763568394002505 \cdot 10^{-15}$	$1.7763568394002489 \cdot 10^{-15}$
8^{-9}	0.0	$2.7755575615628914 \cdot 10^{-17}$
8^{-10}	0.0	$4.336808689942018 \cdot 10^{-19}$
8^{-177}	0.0	$1.012 \cdot 10^{-320}$
8^{-178}	0.0	$1.6 \cdot 10^{-322}$
8^{-179}	0.0	0.0

Tabela 6 wyniki z zadania 7

Wnioski

Bardziej wiarygodna jest funkcja g, ponieważ wartość funkcji powinna zbliżać się do 0, a funkcja g osiąga zero dopiero przy pierwiastku stopnia 179 z 8 (nie jest idealna, ponieważ jednak do tego zera dochodzi, ale jest to spowodowane niedokładnością używanej arytmetyki bardziej niż niedoskonałością funkcji). Funkcja f dochodzi bardzo szybko do zera. Jest to spowodowane odejmowaniem podobnych do siebie liczb. W wyniku takiego działania zawsze dochodzi do utraty cyfr znaczących przez co funkcja ta nie zwraca poprawnych wyników (nawet bliskich).

Zadanie 7

Polecenie

Korzystając ze wzoru $f'(x) \approx \widetilde{f_h'}(x_0) = \frac{f(x_0+h)-f(x_0)}{h}$ obliczyć przybliżoną wartość pochodnej funkcji $f(x) = \sin(x) + \cos(3x)$ w punkcie $x_0=1$ oraz błędów $\left|f'(x_0) - \widetilde{f_h'}(x_0)\right|$ dla $h \in \{2^n : n = 0,1,2,\dots,54\}$

Rozwiązanie

Obliczenie pochodnej ręcznie ($f'(x) = \cos(x) - 3\sin(x)$) i porównanie do przybliżenia otrzymanego z ww. wzoru. Dane przedstawione zostaną w tabelce wraz z 1+h

Wyniki

$$f'(x) = 0.11694228168853815$$

h^{-i}	$\widetilde{f_h'}(x_0)$	$\left\|f'(x_0) - \widetilde{f_h'}(x_0)\right\|$	1+h
2^{-0}	2.0179892252685967	1.9010469435800585	2.0
2^{-1}	1.8704413979316472	1.753499116243109	1.5
2^{-2}	1.1077870952342974	0.9908448135457593	1.25
2^{-3}	0.6232412792975817	0.5062989976090435	1.125
2^{-4}	0.3704000662035192	0.253457784514981	1.0625
…	…	…	…
2^{-25}	0.116942398250103	$1.1656156484463054\ 10^{-7}$	1.0000000298023224
2^{-26}	0.11694233864545822	$5.6956920069239914\ 10^{-8}$	1.0000000149011612
2^{-27}	0.11694231629371643	$3.460517827846843\ 10^{-8}$	1.0000000074505806
2^{-28}	0.11694228649139404	$4.802855890773117\ 10^{-9}$	1.0000000037252903
2^{-29}	0.11694222688674927	$5.480178888461751\ 10^{-8}$	1.0000000018626451
2^{-30}	0.11694216728210449	$1.1440643366000813\ 10^{-7}$	1.0000000009313226
…	…	…	…
2^{-48}	0.09375	0.023192281688538152	1.0000000000000036
2^{-49}	0.125	0.008057718311461848	1.0000000000000018
2^{-50}	0.0	0.11694228168853815	1.0000000000000009
2^{-51}	0.0	0.11694228168853815	1.0000000000000004
2^{-52}	-0.5	0.6169422816885382	1.0000000000000002
2^{-53}	0.0	0.11694228168853815	1.0
2^{-54}	0.0	0.11694228168853815	1.0

Wnioski

Wyniki pokazują, że niektóre obliczenia dawały zbyt małe wyniki, w wyniku czego przybliżenie
zacierało informacje przekazywane przez niektóre zmienne. Na przykład 1+h dla ostatnich iteracji nie różniło się niczym od zwykłej jedynki, ponieważ precyzja float64 nie obejmuje liczby $1+2^{-53}$. Widzimy że dla $h= 2^{-29}$ osiągamy najmniejszy błąd przybliżenia pochodnej, a coraz mniejsze wartości h sprawiają, że błąd znowu rośnie, spowodowane jest to tym że podczas przybliżania pochodnej dzielimy wynik dodawania który wraz z malejącym h praktycznie przestaje się zmieniać (funkcja rośnie wolno, a 1+h = 1 dla małego h), dlatego na końcowych iteracjach błąd jest równy pochodnej, ponieważ przybliżenie jest równe zeru.