Obliczenia Naukowe - Lista 1

Paweł Dychus

Październik 2019

Zadanie 1

Wyznaczenie i interpretacja:

- *macheps*
  x = 1
  Iteracyjnie zmniejszamy wartość zmiennej x, dzieląc przez 2, aż warunek *macheps* t.że $fl(1.0 + macheps) > 1.0$ przestanie być spełniony.

  Otrzymamy dla kolejnych typów:

Typ	Mój wynik	C++	$\approx$
Float16	0.000977	-	$\approx 2^{-10}$
Float32	1.1920929e-7	1.192092896e-07F	$\approx 2^{-23}$
Float64	2.220446049250313e-16	2.2204460492503131e-016	$\approx 2^{-52}$

  Ponadto, mój wynik pokrył się z wartościami zdefiniowanymi w języku Julia.

  Podane wyniki są ostatnimi, dla których zsumowanie z jedynką i powrotne zaokrąglenie na fl nie ucina mantysy.

  Co się tyczy ϵ (precyzja arytmetyki), *macheps* jest jego maszynowym odpowiednikiem. Dla wzoru na $\epsilon = 0.5\beta^{1-t}$, *macheps* otrzymuje te same (jeżeli to możliwe), lub odpowiednio zbliżone wartości do wartości rzeczywistej ϵ, w zależności od określonego typu maszynowego i **długości rejestru**. Stąd pojawia się rozbieżność między wartościami w C, a w Julia.

- MIN_{SUB}
  eta = 1
  Iteracyjnie zmniejszamy wartość zmiennej eta, dzieląc przez 2, aż warunek *eta* t.że $eta > 0$ przestanie być spełniony.

  Otrzymamy dla kolejnych typów:

Typ	Mój wynik	$\approx$
Float16	6.0e-8	$\approx 2^{-24}$
Float32	1.0e-45	$\approx 2^{-149}$
Float64	5.0e-324	$\approx 2^{-1074}$

  Mój wynik koresponduje z wartościami stałych, zapisanymi w języku Julia.

  Podane wyniki są absolutnym minimum wartości jakie można uzyskać w danym typie. Cecha wynosi c_{min} a mantysa ma tylko jedną jedynkę(w systemie bitowym) na samym końcu. Ponadto wartości eta korespondują ze wzorem na $MIN_{SUB} = 2^{-(t-1)} * 2^{c_{min}}$

  Wszystkie wartości mniejsze są zerem maszynowym w danych typach zmiennoprzecinkowych, kalkulatory naukowe jak i przeglądarka google pokazuje 2^{-1074} jako wartość niezerową, dla potęgi = -1075 wszystkie kalkulatory jakie znalazłem zerowały wartość.

- MIN_{NOR}

Typ	floatmin	$\approx$
Float32	1.1754944e-38	$\approx 2^{-126}$
Float64	2.2250738585072014e-308	$\approx 2^{-1022}$

Funkcje zwracają wartości zgodnie ze wzorem na $MIN_{NOR} = 2^{c_{min}}$

W praktyce jest to najniższa wartość, gdzie liczby można przedstawić w formie znormalizowanej. Dla każdej mniejszej wartości cecha zawsze będzie równa $c_{min} - 1$.

- MAX
x = 1
Iteracyjnie zwiększamy wartość zmiennej x, mnożąc przez 2, aż cecha osiągnie wartość maksymalną $c_{max} = 2^{d-1} - 1$. Następnie wypełniamy mantysę kolejnymi potęgami dwójki, aż do wypełnienia jej jedynkami. Otrzymamy dla kolejnych typów:

Typ	Mój wynik	C++	<
Float16	6.55e4	-	$< 2^{16}$
Float32	3.4028235e38	3.402823466e+38F	$< 2^{127}$
Float64	1.7976931348623157e308	1.79769e+308	$< 2^{1023}$

Mój wynik koresponduje z wartościami stałych, zapisanymi w języku Julia.

Podane wyniki są **największymi** wartościami, cecha wynosi c_{max}, a mantysa jest zapełniona jedynkami. Następna wartość to już ∞.

Dla tak wielkich wartości operacje arytmetyczne są obdarzone ogromnymi błędami, widać to po stopniu w potędze dwójki kolejnych liczb, którymi wypełnialiśmy mantysę. Dodawanie niewielkich wartości nie ma w takim przypadku żadnego sensu.

Zadanie 2

Czy i dlaczego metoda Kahana dla wyliczania machepsa działa?

Dla kolejnych typów T:

```
x = 3(T(4)/3-1) - 1
println(x)
```

Dostajemy wartości równe zdefiniowanym -eps(T) w Julii, z zastrzeżeniem Float32,
gdzie wartość równa się eps(Float32).

Główna magia tego działania opiera się na operacji odejmowania zaraz po dzieleniu, to tam mantysa traci dokładność z prostego powodu, że okres (01) nie jest pamiętany i wstawiane zostają 00. W efekcie po przemnożeniu przez 3, otrzymujemy wartość minimalnie różną od 1 i po ponownym odejmowaniu, naszego machepsa.
Jeżeli zaś chodzi o ujemnego eps dla Float32, spowodowane jest to tym, że podczas operacji dzielenia, dokładniej aproksymacji, t+1 ma wartość 1, a nie 0 jak w przypadku Float16 i Float64. Przez to końcowa wartość jest minimalnie większa od 1 zamiast mniejsza.

Jak widać operacje potrafią wprowadzić w błąd, już bardzo proste działanie sprawia, że sprawdzenie czy wartość jest istotnie równa 0 nie ma sensu.
A sposobów na wyznaczenie machepsa prawdopodobnie jest więcej.

Zadanie 3

Jakie jest rozmieszczenie liczb zmiennoprzecinkowych, ile ich jest i w jakich przedziałach.

Ze względów optymalizacyjnych(algorytm o 2^{52} ilości iteracji), eksperymentalne rozwiązanie sprowadzimy do wyświetlenia bloku bitów. 3 razy dla stanu początkowego i stan ostatni.
delta - współczynnik rozmieszczenia

- Na przedziale liczb [1, 2] dla delty równej 2^{-52}

Iteracje	Bitstring
1	0011111111110000000000000000000000000000000000000000000000000001
2	0011111111110000000000000000000000000000000000000000000000000010
...	
$2^{52}-1$	0011111111111111111111111111111111111111111111111111111111111111

- Na przedziale liczb [0.5, 1] dla delty równej 2^{-53}

Iteracje	Bitstring
1	0011111111100000000000000000000000000000000000000000000000000001
2	0011111111100000000000000000000000000000000000000000000000000010
...	
$2^{52}-1$	0011111111101111111111111111111111111111111111111111111111111111

- Na przedziale liczb [2, 4] dla delty równej 2^{-51}

Iteracje	Bitstring
1	0100000000000000000000000000000000000000000000000000000000000001
2	0100000000000000000000000000000000000000000000000000000000000010
...	
$2^{52}-1$	0100000000001111111111111111111111111111111111111111111111111111

Jak można zauważyć dla przypadku [2, 4] cecha zwiększyła się, a dla [0.5, 1] zmniejszyła się. Zmiana cechy wpłynęła również na zmianę wartości delty. Zastosowanie delty równej 2^{-52} dla przypadku [0.5, 1] nie zmieniło by wartości.
Na tej podstawie łatwo zauważyć, że delta wynosi: 2^{-52+c} na kolejnych przedziałach

Innymi słowy, dla kolejnych przedziałów: $\{2^k[1,2] : k \in [c_{min}, c_{max}]\}$, ilość elementów w tym przedziale jest stała, zmienia się jedynie współczynnik rozmieszczenia - delta.

Zadanie 4

1. Wyznaczyć liczbę x: $(1 < x < 2)$ że $(x * (1/x) \neq 1)$

   Iterujemy w dół o wartość 2^{-52} dla początkowego x = 2.

   Otrzymujemy
   x:1.9999999850988384
   result: 0.9999999999999999

   Rezultat, jest efeketem niedokładności odwracania, widać to bardziej na przykładzie 3 i 1/3, która ma okres(01) w zapisie binarnym. Jest to również bliźniacze zadanie, do zadania 2, gdzie wyznaczaliśmy *machepsa* metodą Kahana.

2. Wyznaczyć min(x).

   Bierzemy wartość minimalną:MIN_{SUB} i iterujemy w górę.

   Otrzymujemy
   x:5.0e-324
   result: ∞
   Rezultat, raczej oczywisty, cecha osiąga wartość maksymalną, przez co dostajemy ∞.

Ze względu na skończoną dokładność, warunek $(x * (1/x) = 1)$ jak i inne proste operacje, mogą być w niektórych przypadkach nieprawdziwie, względem rzeczywistych wartości.

Zadanie 5

Obliczyć iloczyn skalarny wektorów
$x = [2.718281828, -3.141592654, 1.414213562, 0.5772156649, 0.3010299957]$
$y = [1486.2497, 878366.9879, -22.37492, 4773714.647, 0.000185049]$.
Czterema sposobami:

1. W przód, poczynając od dodawania pierwszych indeksów

Typ	Wynik
Float64	1.0251881368296672e-10
Float32	-0.4999443

2. W tył, poczynając od dodawania ostatnich indeksów

Typ	Wynik
Float64	-1.5643308870494366e-10
Float32	-0.4543457

3. Od największego do najmniejszego

Typ	Wynik
Float64	0.0
Float32	-0.5

4. Od najmniejszego do największego

Typ	Wynik
Float64	0.0
Float32	-0.5

Najbliżej wartości(odległość do poprawnego wyniku) okazało się być:

- Pierwsze miejsce, algorytm 3 i 4 dla Float 64: 1.0065710700000004e-11.
- Na drugim miejscu, algorytm 1 dla Float 64: 1.1258452438296672e-10.
- Na trzecim, algorytm 2 dla Float 64: 1.4636737800494365e-10
- Float 32 był porównywalnie beznadziejny.

Patrząc po wynikach, operacje na zbiorze posortowanym tj. w kolejności max-min i vice versa dają najbliższe prawdy rezultaty.

Zadanie 6

Dla kolejnych wartości x = $8^{-1}, 8^{-2}, 8^{-3}, 8^{-4}, ...$ Obliczyć wartość i wiarygodność(precyzję) danych funkcji:

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

Otrzymujemy następujące wyniki:

i	$f(8^{-i})$	$g(8^{-i})$
1	0.41421356237309515	0.4142135623730951
2	0.0077822185373186414	0.0077822185373187065
3	0.00012206286282867573	0.00012206286282875901
...		
9	1.7763568394002505e-15	1.7763568394002489e-15
10	0.0	2.7755575615628914e-17
11	0.0	4.336808689942018e-19
...		
189	0.0	1.6e-322
190	0.0	0.0

Jak widać, już dla i = 10, f(x) traci wartość. W przeciwieństwie do g(x), które zwraca wartości aż dla indeksu 189, gdzie wartość jest bliska MIN_{SUB}.

Cała magia tyczy się dodawania liczby 1. W f(x), dla $i = 10$, otrzymujemy wartość: $x < macheps$ ($2^{-60} < 2^{-52}$) z definicji otrzymując pod pierwiastkiem 1. Po odjęciu dostajemy 0. Podczas gdy, w g(x) mamy dzielenie $x^2/2$.

Wniosek jest prosty, dobór funkcji z priorytetem na unikanie operacji mogących szybko zerować wynik, jest bardzo istotne, jeżeli chcemy otrzymać precyzyjne wartości.

Zadanie 7

Wyznaczyć wartości ze wzoru na pochodną: $f^{`}(x_0) = \frac{f(x_0+h)-f(x_0)}{h}$ i porównać wartości z rzeczywistą wartością pochodnej, dla $h = 2^{-n}, n \in [0, 1, ..., 54]$ w punkcie $x_0 = 1$.
$f(x) = sinx + cos3x$, $f^{`}(1) = 0.11694228168853815$
Otrzymujemy poniższe wyniki:

h	1+h	pochodna	błąd
2^{-0}	2.0	2.0179892252685967	1.9010469435800585
2^{-1}	1.5	1.8704413979316472	1.753499116243109
2^{-2}	1.25	1.1077870952342974	0.9908448135457593
2^{-3}	1.125	0.6232412792975817	0.5062989976090435
2^{-4}	1.0625	0.3704000662035192	0.253457784514981
2^{-5}	1.03125	0.24344307439754687	0.1265007927090087
2^{-6}	1.015625	0.18009756330732785	0.0631552816187897
2^{-7}	1.0078125	0.1484913953710958	0.03154911368255764
2^{-8}	1.00390625	0.1327091142805159	0.015766832591977753
2^{-9}	1.001953125	0.1248236929407085	0.00788141125217034
...			
2^{-19}	1.0000019073486328	0.11694997636368498	7.694675146829866e-6
2^{-20}	1.0000009536743164	0.11694612901192158	3.8473233834324105e-6
2^{-21}	1.0000004768371582	0.1169442052487284	1.9235601902423127e-6
2^{-22}	1.000000238418579	0.11694324295967817	9.612711400208696e-7
2^{-23}	1.0000001192092896	0.11694276239722967	4.807086915192826e-7
2^{-24}	1.0000000596046448	0.11694252118468285	2.394961446938737e-7
2^{-25}	1.0000000298023224	0.116942398250103	1.1656156484463054e-7
2^{-26}	1.0000000149011612	0.11694233864545822	5.6956920069239914e-8
2^{-27}	1.0000000074505806	0.11694231629371643	3.460517827846843e-8
2^{-28}	1.0000000037252903	0.11694228649139404	**4.802855890773117e-9**
2^{-29}	1.0000000018626451	0.11694222688674927	5.480178888461751e-8
2^{-30}	1.0000000009313226	0.11694216728210449	1.1440643366000813e-7
2^{-31}	1.0000000004656613	0.11694216728210449	1.1440643366000813e-7
2^{-32}	1.0000000002328306	0.11694192886352539	3.5282501276157063e-7
2^{-33}	1.0000000001164153	0.11694145202636719	8.296621709646956e-7
2^{-34}	1.0000000000582077	0.11694145202636719	8.296621709646956e-7
2^{-35}	1.0000000000291038	0.11693954467773438	2.7370108037771956e-6
...			
2^{-48}	1.0000000000000036	0.09375	0.023192281688538152
2^{-49}	1.0000000000000018	0.125	0.008057718311461848
2^{-50}	1.0000000000000009	0.0	0.11694228168853815
2^{-51}	1.0000000000000004	0.0	0.11694228168853815
2^{-52}	1.0000000000000002	-0.5	0.6169422816885382
2^{-53}	1.0	0.0	0.11694228168853815
2^{-54}	1.0	0.0	0.11694228168853815

Najlepszą aproksymację osiągnęliśmy dla potęgi -28. Dla kolejnych potęg precyzja zaczyna maleć. Jest to spowodowane zjawiskiem redukcji cyfr znaczących, która dla dostatecznie bliskich liczb, wprowadzają wiele zer nie noszących informacji.
Operacja 1+h zachowuje się w przewidywalny sposób i dla h będącego -52 potęgą dwójki stanowi *macheps*

Konkluzja, to unikać odejmowania liczb bardzo bliskich siebie, ponieważ prowadzi to do redukcji cyfr znaczących.