Pracownia z analizy numerycznej

Sprawozdanie do zadania **P1.3**

Prowadzący pracownię: Paweł Woźny

Łukasz Klasiński

Wrocław, 11 listopada 2017

1 Wstęp

Sumowanie liczb jest problemem pojawiającym się niemal na każdym kroku. Używają go niemalże wszystkie algorytmy i jest czymś tak normalnym, że często zapomina się jakie mogą się za nim kryć niebezpieczeństwa. Kiedy ktoś potrzebuje zaimplementować dodawanie jakiejś sumy liczb, zwykle sięgnie po zwykłe, "naiwne" dodawanie "po kolei". I przy typowych danych wynik, rzeczywiście jest zgodny z oczekiwaniami. Ale jak później pokażę przy odpowiednich warunkach oraz danych wejściowych dodawanie może zwracać bardzo zniekształcone wyniki.

Głównym celem niniejszego sprawozdania będzie sprawdzenie, jak zachowuje się obliczanie wartości sumy

$$\sum_{k=1}^{n} a_k \text{ gdzie } n := 2^m \text{ dla pewnego } m \in \mathbb{N} \tag{1}$$

dla odpowiednich danych. Poza tym zostaną przedstawione inne sposoby sumowania liczb, które mają lepszą poprawność numeryczną.
DODAĆ NAWIAZANIA DO ALGORYTMÓW!!!
Wszystkie testy numeryczne przeprowadzono przy użyciu języka programowania **Julia v.0.6.1** w trybie pojedyńczej (**Single**) oraz podwójnej (**Double**) precyzji, czyli kolejno 32 oraz 64 bitowej dokładności.

2 Uwarunkowanie sumowania liczb

Najprostszym sposobem na znalezienie danych, które mogą mocno zaburzyć wynik, jest wyliczenie wskaźnika uwarunkowania zadania. Korzystając ze

wzoru

$$|\frac{f(x+h)-f(x)}{f(x)}| \tag{2}$$

Nasza funkcja sumowania to inaczej

$$f(x_1, x_2, x_3, \ldots, x_n) = x_1 + x_2 + \ldots + x_n$$

Przyjmijmy

$$\ddot{x_i} := x_i(1+\xi_i), \text{ gdzie } |\xi_i| < \epsilon$$

Wówczas

$$|\frac{\ddot{f}-f}{f}| = |\frac{f(\ddot{x_1}, \ddot{x_2} \ldots \ddot{x_n}) - f(x_1, x_2 \ldots x_n)}{f(x_1, x_2 \ldots x_n)}| =$$

$$|\frac{\sum_i^n x_i + x_i\xi_i - \sum_i^n x_i}{\sum_i^n x_i}| = |\frac{\sum_i^n x_i\xi_i}{\sum_i^n x_i}| \leq \frac{\sum_i^n |x_i|}{|\sum_i^n x_i|}\delta$$

Wtedy δ jest względną zmianą danych, a $\frac{\sum_i^n |x_i|}{|\sum_i^n x_i|}$ wskaźnikiem uwarunkowania zadania.

Z wskaźnika uwarunkowania wynika, że gdy $\sum_i^n |x_i|$ jest duża, a $|\sum_i^n x_i|$ mała, to zadanie jest źle uwarunkowanie, ponieważ przy niewielkiej zmianie danych wynik będzie zupełnie inny.

3 Poprawność numeryczna

Zobaczmy, jak wygląda poprawnośc numeryczna tego problemu. Pokaże to słabe strony "naiwnego" algorytmu sumowania.

Załóżmy, że $a_1, a_2, a_3, \ldots, a_n$ są liczbami maszynowymi, a u precyzją arytmetyki. Wówczas

$$fl(\sum_{k=1}^{n} a_k) =$$

$$= fl(a_1 + a_2 + a_3 + \ldots + a_n) =$$

$$= fl(fl(a_1 + a_2 + a_3 + \ldots + a_{n-1}) + a_n) =$$

$$= fl(fl(fl(a_1 + a_2 + a_3 + \ldots + a_{n-2}) + a_{n-1}) + a_n) =$$

$$= fl(fl(fl(\ldots fl(a_1 + a_2) + a_3) + \ldots)\ldots) + a_n) =$$

$$= fl(fl(fl(\ldots (a_1 + a_2)(1+\xi_2) + \ldots)\ldots) + a_n), \text{ gdzie } |\xi_i| \leq u$$

$$= fl(fl(fl(\ldots((a_1 + a_2)(1 + \xi_2) + a_3)(1 + \xi_3)\ldots) + a_n) =$$

$$= (\ldots((a_1 + a_2)(1 + \xi_2) + a_3)(1 + \xi_3)\ldots) + a_n)(1 + \xi_n) =$$

$$= \ddot{a_1} + \ddot{a_2} + \ldots + \ddot{a_n}$$

$$\text{gdzie } \ddot{a_i} = a_i * (1 + \delta_i),\ \delta_i = \prod_i^n (1 + \xi_i) \text{ dla } \xi_1 = 0$$

$$\text{wtedy } |\delta_i| \leq i * u$$

Stąd widać, że algorytm jest numerycznie poprawny. Ale poza tym daje nam ważną informację - widać, że kumulacja błędu jest największa przy pierwszych dwóch wyrazach i z każdym kolejnym jest o 1 mniejsza.

4 Sumowanie binarne

Ponieważ w (1) nasze n jest potęgą 2, to naturalnym wydaje się zastosowanie dodawania binarnego. W praktyce różni się to od "naiwnego" dodawania rozmieszczeniem nawiasów. O ile wcześniej dodawanie wyglądało

$$(\ldots(((a_1 + a_2) + a_3) + a_4)\ldots) + a_n)$$

Tak w dodawaniu binarnym nawiasowanie wygląda inaczej

$$(\ldots\{[(a_1 + a_2) + (a_3 + a_4)] + [(a_5 + a_6) + (a_7 + a_8)]\}\ldots(a_{n-1} + a_n)]\}\ldots)$$

Tak rozmieszczone nawiasy powodują jednak, że błąd obarczający zmienne podczas dodawania jest zupełnie inaczej rozmieszczony.

4.1 Poprawność numeryczna sumowania binarnego

Tak jak wcześniej załóżmy, że $a_1, a_2 \ldots a_n$ są liczbami maszynowymi, a u precyzją arytmetyki oraz $n = 2^m$ takie, że $m \in \mathbb{N}$. Wówczas

$$fl(bSum(a_1, a_2 \ldots a_n)) =$$

$$= fl([(a_1 + a_2) + (a_3 + a_4)] + \ldots) =$$

$$= fl(fl(\ldots fl[fl(a_1 + a_2) + fl(a_3 + a_4)] + \ldots)$$

$$= (\ldots([(a_1+a_2)(1+\xi_1)+(a_2+a_3)(1+\xi_1')](1+\xi_2)\ldots)(1+\xi_m), \text{ gdzie } |\xi_i| \leq u =$$

$$= \ddot{a_1} + \ddot{a_2} + \ddot{a_3} + \ldots + \ddot{a_n}, \text{ gdzie } \ddot{a_i} = a_i(1 + \xi_1)(1 + \xi_2)(1 + \xi_3)\ldots(1 + \xi_n)$$

$$\text{Zatem } \ddot{a_i} = a_i(1+\delta_i), \text{ przy czym } \delta_i = \prod_i^m (1+\xi_i)$$

$$\text{Ostatecznie } |\delta_i| \leq mu$$

Zatem algorytm jest numerycznie poprawny. Widać zatem, że w tym przypadku rozmieszczenie błędu jest o wiele bardziej równomierne.

5 Algorytm Kahana

Algorytm Kahana pozwala znacząco zredukować błąd numeryczny uzyskiwany poprzez sumowanie. Jest to możliwe dzięki konsekwentemu wyliczaniu po każdym sumowaniu błędu i dodawaniu go do wyniku aby go zredukować. Aby zrozumieć jego działanie zobatrzmy przykład:
Chcemy dodać dwie liczby w środowisku, gdzie mamy ograniczoną precyzję. Załóżmy, że używamy sześcio-cyfrowej dziesiętnej arytmetyki zmiennoprzecinkowej. Wtedy, gdy chcemy dodać trzy zmienne x, y, z takie, że

$$x := 10000.0,\ y := 3.14159,\ z := 2.71828$$

Wtedy naiwne dodanie tych zmienny wyniesie

$$x + y = 10003.14159$$

co przez precyzję zostanie zaokrąglone do 10003.1

$$10000.1 + z = 10005.81828 \approx 10005.8$$

Ale przecież

$$10000.0+3.14159+2.71828 = 10003.14159+2.71828 = 10005.85987 \approx 10005.9$$

Zatem błąd wygenerowany przy dodawaniu x i y spowodował, że ostateczny wynik jest inny od oczekiwanego. Spróbujmy to naprawić.
Zauważmy, że jeśli obliczmy $((x + y) - x) - y$, to wynik powinien zawsze wynosić 0. Ale ponieważ mamy ograniczoną precyzję, to

$$((10000.0 + 3.14159) - 10000.0) - 3.14159 \approx (10003.1 - 10000.0) - 3.14159 =$$

$$= 3.10000 - 3.14159 \approx -0.04159$$

Jest wartość, którą utraciliśmy przez precyzję. Teraz, gdy dodamy ją do z i wykonamy sumę, to otrzymamy

$$z + 0.04159 = 2.75987$$

$$10003.1 + 2.75887 = 10005.85987 \approx 10005.9$$

Dzięki temu pomimo ograniczonej precyzji jesteśmy w stanie przy dużej ilościu sum znacząco zredukować błąd. W poniższych eksperymentach został wykorzystany nieco ulepszony algorytm Kahana nazywany "improved Kahan-Babuska algorithm", który różni się tym, że błąd kumuluje w oddzielnym liczniku i dodaje na końcu. Więcej o algorytmie Kahan-Babuska oraz jak go ulepszyć można znaleźć w [1].

6 Testy i analiza

7 Podsumowanie

Literatura

[1] A. Klein, A generalize Kahan-Babuska-Summation-Algorithm, 2005.