

Porównanie algorytmu sumacyjnego Kahana z algorytmem naiwnym (**zad. P1.2**)

Paweł Laskoś

grupa dr Pawła Woźnego, nr indeksu 169827

6 listopada 2005

1 Wstęp

Celem eksperymentu było zbadanie wpływu zjawiska utraty cyfr znaczących na otrzymane wyniki sumy wyrazów ciągu w zależności od wybranego algorytmu sumacyjnego. Na przykładzie sumy

$$\sum_{k=1}^{10000} x_k, \text{ gdzie } x_k = k^{-2} \quad (1)$$

porównano dwa algorytmy sumacyjne: naiwny

```
s := x1;  
for i from 2 to n do  
s := s + xi;  
end
```

(zastosowany w arytmetykach pojedynczej i podwójnej precyzji) oraz przypisywany Williamowi Kahanowi algorytm *sumowania z poprawkami*

```
s := x1;  
c := 0;  
for i from 2 to n do  
y := c + xi;  
t := s + y;  
c := (s - t) + y;  
s := t;  
end
```

zwany także wzorem sumacyjnym Kahana, sumowaniem kompensacyjnym itd., zastosowany w arytmetyce pojedynczej precyzji.

2 Opis teoretyczny

Niech \bar{s} oznacza sumę wyrazów x_i wyliczoną algorytmem naiwnym. Jej reprezentacja w arytmetyce zmiennoprzecinkowej

$$fl(\bar{s}) = \sum_{k=1}^n x_k(1 + \eta_k), \quad (2)$$

gdzie $|\eta_k| \leq \epsilon_1(n-i+1)$ to suma błędów wynikających z utraty cyfr znaczących podczas dodawania. Epsilon maszynowy definiujemy następująco:

$$\epsilon = 2^{-\tau}, \quad (3)$$

gdzie τ jest liczbą bitów przeznaczonych w arytmetyce fl na mantysę.

Dowód równania (2). Zauważmy, że

$$\begin{aligned} \bar{s} &= fl(\dots fl(fl(x_1 + x_2) + x_3) \dots + x_n) = \\ &= (\dots ((x_1 + x_2)(1 + \sigma_2) + x_3)(1 + \sigma_3) \dots + x_n)(1 + \sigma_n) = \\ &= x_1 \prod_{i=2}^n (1 + \sigma_i) + \sum_{i=2}^n x_i \prod_{j=i}^n (1 + \sigma_j). \end{aligned} \quad (4)$$

Następnie możemy przyjąć $\sigma_1 = 0$ i napisać

$$\bar{s} = \sum_{i=1}^n x_i \prod_{j=i}^n (1 + \sigma_j). \quad (5)$$

Następnie, skoro wszystkie zmiennopozycyjne poprawki dodawania spełniają $|\sigma_i| < \epsilon_1$, na mocy twierdzenia 1.4 [1, s. 2] piszemy

$$\bar{s} = \sum_{i=1}^n x_i (1 + \eta_i), \quad (6)$$

gdzie $\eta_i \approx \sum_{i=1}^n \sigma_i \leq \sum_{i=1}^n |\sigma_i| \leq (n-i+1)\epsilon_1$. □

Niech s oznacza sumę wyrazów x_i wyliczoną algorytmem Kahana. Zachodzi fakt

$$fl(s) = \sum_{k=1}^n x_k (1 + \delta_k) + O(n\epsilon_2^2) \sum_{k=1}^n |x_k|, \quad (7)$$

gdzie $|\delta_k| \leq 2\epsilon_2$. Dowód tego faktu cytowany jest za [2, s. 572 i nast.] w [3, ss. 246-248].

3 Wyniki doświadczenia

Załączony kod `kahan.c` implementuje omówione algorytmy w języku ANSI C. Tabela 1 przedstawia z dokładnością do 20 cyfr dziesiętnych wartości zmiennych po zakończeniu działania programu, gdzie

fl zawiera wartość wyliczoną algorytmem naiwnym w arytmetyce pojedynczej precyzji `float`,

dbl – wartość wyliczoną algorytmem naiwnym w arytmetyce podwójnej precyzji `double`,

kahan – wartość wyliczoną algorytmem sumacyjnym Kahana w arytmetyce pojedynczej precyzji,

maple – załączoną dla porównania wartość wyliczoną przez program *Maple 9.5*.

W tabeli podkreślono cyfry znaczące zgodne z wynikiem obliczeń *Maple*.

zmienna	wartość
<i>fl</i>	1,66472532272338867188
<i>dbl</i>	1,64483407184806496026
<i>kahan</i>	1,66483404159545898438
<i>maple</i>	1,64483407184805985324

Tabela 1: Wyniki obliczeń

4 Wnioski

Jeśli w równaniach (2,7) przyjmiemy

$$\epsilon_1 = \epsilon_2 = \epsilon_{fl}, \quad (8)$$

gdzie ϵ_{fl} to epsilon maszynowy w arytmetyce z pojedynczą precyzją, to można z nich wywnioskować, że dla dużych n błąd przybliżenia sumy proporcjonalny jest do ϵ_{fl} dla algorytmu naiwnego i do ϵ_{fl}^2 dla algorytmu z poprawkami. Potwierdzają to wyniki doświadczalne – jeśli przyjmiemy wynik obliczony przez program *Maple* za wartość dokładną, to wartość wyliczona algorytmem naiwnym jest zgodna z nią na 4 cyfrach wiodących, zaś algorytmem z poprawkami – na 8.

Spróbujmy teraz porównać wyniki działania algorytmu z poprawkami i algorytmu naiwnego, ale w arytmetyce z podwójną precyzją. Specyfikacja IEEE 754 mówi, że liczba bitów poświęconych na mantysę w arytmetyce z podwójną precyzją jest dwukrotnie większa, niż w arytmetyce z pojedynczą precyzją, tj.

$$\tau_{dbl} = 2\tau_{fl}. \quad (9)$$

Ponadto wykorzystuje się fakt, że wiodący bit mantysy znormalizowanej (w systemie dwójkowym) zawsze ma wartość 1, i epsilon maszynowy wynosi wtedy

$$\epsilon = 2^{-\tau-1}, \quad (10)$$

nie zaś tyle, ile wskazuje równanie (3). Okazuje się wtedy, że zachodzi zależność

$$\epsilon_{fl}^2 = 2^{-2\tau_{fl}-2} = \frac{1}{2}2^{-\tau_{dbl}-1} = \frac{\epsilon_{dbl}}{2}, \quad (11)$$

gdzie ϵ_{dbl} to epsilon maszynowy w arytmetyce z podwójną precyzją. Na tej podstawie możnaby wnosić, że przyjęcie

$$\epsilon_1 = \epsilon_{dbl} = 2\epsilon_{fl}^2, \quad \epsilon_2 = \epsilon_{fl} \quad (12)$$

implikuje porównywalną zgodność wyników algorytmu z poprawkami i algorytmu naiwnego w arytmetyce z podwójną precyzją z wartością dokładną. Jak jednak widać w tabeli 1, wynik algorytmu naiwnego w arytmetyce z podwójną precyzją jest daleko bliższy dokładnemu wynikowi, niż wynik algorytmu z poprawkami. Zauważmy jednak, że epsilon maszynowy w arytmetyce z pojedynczą precyzją (zgodnie ze standardem IEEE, który przyjmuje $\tau_{fl} = 23$) wynosi

$$\epsilon_{fl} = 2^{-23-1} \approx 5,960464 \times 10^{-8} \approx 0,00000006, \quad (13)$$

a zatem zachodzi

$$|maple - kahan| < \epsilon_{fl}, \quad (14)$$

co oznacza, że dla wybranego w zadaniu x_i , n algorytm z poprawkami wylicza najdokładniejszą możliwą do wyrażenia w arytmetyce z pojedynczą precyzją wartość sumy $\sum_{k=1}^n x_i$.

Literatura

- [1] Stanisław Lewanowicz, *Podstawowe pojęcia teorii błędów w analizie numerycznej*, notatki do wykładu z analizy numerycznej, 2005¹.
- [2] Donald E. Knuth, *The Art of Computer Programming*, Volume 2, 2nd Ed., Addison-Wesley, 1981.
- [3] David Goldberg, *What Every Computer Scientist Should Know About Floating-Point Arithmetic*, Association for Computing Machinery, Inc., 1991².

¹<http://www.ii.uni.wroc.pl/~sle/an-tbl.pdf>

²<http://www.validlab.com/goldberg/paper.pdf>