Analiza Numeryczna (M) - Pracownia 1 - Zadanie P1.2

Sumowanie liczb zmiennopozycyjnych

Prowadzący: dr hab. Paweł Woźny

Karolina Jeziorska

Wrocław, 29 grudnia 2019

Spis treści

1. Wstęp

Dodawanie jest podstawym działaniem arytmetycznym. Dlatego chielibyśmy żeby było jak najdokładniejsze i pozbawione błędów - nawet w komputerze. Jednak w arytmetyce zmiennopozycyjnej każde działanie obarczone jest błędem. A czym więcej działań, tym możliwie większy błąd. Dlatego obliczona suma ciagu liczb zmiennopozycyjnych może się znacznie różnić od dokładnego wyniku. Jednak ten błąd nie jest zawsze taki sam - zależy od ilości liczb, ale również od ich wielkości, użytej precyzji czy nawet zastosowanego algorytmu.

Głównym celem niniejszego sprawozdania jest analiza i porównanie różnych metod sumowania liczb zmiennopozycyjnych, aby móc określić, które dają najdokładniejsze wyniki. Przyjrzymy się dokładniej zwykłemu sumowaniu (opisanemu w rozdziale **??**) oraz algorytmowi sumacyjnemu Kahana (przedstawionemu w rozdziale **??**), a następnie przetestujemy oba algorytmy w pojedynczej i podwójnej precyzji (rozdział **??**).

2. *Normalne* sumowanie

2.1. Analiza algorytmu

Naszym celem jest obliczenie $\sum_{i=1}^{n} x_i$ - sumy n liczb rzeczywistych. Można to wykonać za pomocą podstawowego algorytmu poprzez dodawanie kolejnych liczb ciągu x_i.

```
s:=0
for i from 1 to n
do
    s:=s+x[i]
end
```

2.2. Analiza poprawności

Dość łatwo oszacować błąd tego algorytmu

Niech $S_n = \sum_{i=1}^{n} x_i$ i niech $\hat{S_n} = fl(\sum_{i=1}^{n} x_i)$

Mamy wtedy: (dla $\delta_1 = 0$)

$\hat{S_n} = fl(x_1 + x_2 + \cdots + x_n) = ((((x_1(1+\delta_1) + x_2)(1+\delta_2) + x_3)(1+\delta_3) + \cdots + x_n)(1+\delta_n) =$

$x_1(1+\delta_1)(1+\delta_2)\ldots(1+\delta_n) + x_2(1+\delta_2)\ldots(1+\delta_n) + \cdots + x_n(1+\delta_n)$

Gdzie $\delta_i \leqslant \frac{u}{1-u}$ a z ćwiczeń wiemy, że $\prod_{i=1}^{k}(1+\delta_i) = 1+\gamma_k$ gdzie $\gamma_k \leqslant \frac{ku}{1-ku} = \theta_k$

Czyli mamy $\hat{S_n} = x_1(1+\gamma_n) + x_2(1+\gamma_{n-1}) + \cdots + x_n(1+\gamma_1) = \sum_{i=1}^{n} x_i(1+\gamma_{n-i+1})$

Wtedy $|E_n| = |S_n - \hat{S_n}| = |\sum_{i=1}^{n} x_i\gamma_{n-i+1}| \leqslant \sum_{i=1}^{n} |x_i|\theta_{n-i+1} \leqslant \theta_n \sum_{i=1}^{n} |x_i|$

(1)

Jeśli spojrzymy na pierwsze ograniczenie, zauważymy że zależy od kolejności sumowania - przy pierwszym elemencie sumy kumuluje się największy błąd (można go ograniczyć przez $\frac{nu}{1-nu}$), który przy kolejnych elementach jest coraz mniejszy (finalnie przy ostatnim elemencie można już go tylko ograniczyć przez $\frac{u}{1-u}$). Stąd łatwo wnioskować, że w celu zminimalizowania błędu należy dodawać elementy w kolejności rosnącej, tak by największy błąd stał przy elemencie najmniejszym, a najmniejszy błąd przy największym. Trzeba jednak zwrócić uwagę, że bierzemy moduł z elementu stojącego przy błędzie, to znaczy, że ciąg, który sumujemy należy sortować co do wartości modułu elementów.

Warto również zauważyć, że gdy szacujemy błąd względny, to można go (niezależnie od kolejności sumowania) ograniczyć z góry przez

$$\frac{\theta_n \sum_{i=1}^{n} |x_i|}{|\sum_{i=1}^{n} x_i|}$$

Jeżeli wszystkie x_i są tego samego znaku - jest to po prostu θ_n. Jednak, jeżeli elementy ciągu są różnych znaków, nie jesteśmy w stanie ograniczyć błędu względnego.

3. Algorytm sumacyjny Kahana

Algorytm sumacyjny Kahana (nazywany inaczej algorytmem sumowania z poprawkami) pozwala minimalizować błędy, które pojawiają się przy sumowaniu liczb zmiennopozycyjnych w skończonej precyzji. Za twórcę algorytmu uważa się Williama Kahana.

3.1. Analiza algorytmu

Algorytm zawiera więcej operacji niż algorytm przedstawiony w rodziale **??**. Te dodatkowe operacje pozwalają uniknąć utraty mniej znaczacych bitów, gdy dodajemy do liczby względnie do niej mniejszą. Przeanalizujmy algorytm Kahana:

```
suma:=x[1]
c:=0
for i from 2 to n
do
    y:=c+x[i]
    t:=suma+y
    c:=(suma-t)+y
    suma:=t
end
```

Zmienna c to poprawka, która zawiera bity utracone podczas dodawania. Na początku wynosi 0, bo suma to dokładnie pierwszy element ciągu. W pętli wartość zmiennej suma może być względnie duża w stosunku do wartości zmiennej y, co powoduje utratę bitów mniej znaczących zmiennej y. W kolejnym kroku operacja (suma-t)+y odzyskuje *zgubione* niższe bity zmiennej y i zapisuje je w zmiennej c, by w kolejnej iteracji utracone bity zmiennej y zostały dodane do następnego elementu ciągu.

3.2. Analiza poprawności

Z powodu większej ilości operacji, trudniej jest oszacować błąd algorytmu Kahaha. Wiemy jednak, że:

$$\begin{aligned}&\text{Niech } S_n = \sum_{i=1}^{n} x_i \text{ i niech } \hat{S_n} = fl(\sum_{i=1}^{n} x_i) \text{ i } \hat{S_n} \text{ zostało wyliczone za pomocą algorytmu Kahana}\\&\text{Wtedy } \hat{S_n} = \sum_{i=1}^{n}(1+\xi_i)x_i \text{ gdzie } |\xi_i| \leqslant 2 \cdot 2^{-t} + O(n2^{-2t})\end{aligned} \tag{2}$$

Wykazanie tej zależności przebiega dokładnie tak jak poprzednio, chcemy wyliczyć błąd przy x_i. Jednak w tym wypadku łatwiej na to spojrzeć rekurencyjnie - $\hat{s_1} = x_1(1+\delta_1)$ $\hat{s_l} = (\hat{s_{l-1}}+x_l)(1+\delta_l)$ (gdzie $\hat{s_l} = fl(\sum_{i=1}^{l} x_i)$), wtedy współczynniki przy x_1 to $(1+\delta_1)(1+\delta_2)\dots(1+\delta_n)$, a przy x_2 to $(1+\delta_2)\dots(1+\delta_n)$ - jak już wiemy, błąd przy x_1 jest największy i można nim szacować pozostałe błędy. Wygląda to tak samo w przypadku algorytmu Kahana, jednak współczynnik przy x_1 jest trochę bardziej skomplikowany. Spójrzmy więc na wszystkie operacje wykonywane w algorytmie:

$y_k = (x_k + c_{k-1})(1 + \eta_k)$
$s_k = (s_k + y_k)(1 + \alpha_k)$
$c_k = ((s_{k-1} - s_k)(1 + \gamma_k) + y_k)(1 + \beta_k)$
gdzie wszystkie greckie litery są ograniczone przez $u = 2^{-t}$
Chcemy obliczyć współczynnik (błąd), który stoi przy x_1, w tym celu wyliczamy S_k,
czyli współczynniki przy x_1 w s_k gdzie s_k to suma k elementów ciągu. (3)
Pomijając obliczenia, które dokładne można znaleźć w [**?**] mamy
$S_k = 1 + \eta_1 - \gamma_1 - (4k + 1)u^2 + O(u^3)$
To, co finalnie chcemy, to współczynniki przy x_1 w s_n, czyli
$S_n = 1 + 2u + O(nu^2)$
czyli $|\xi_i| \leqslant 2 \cdot 2^{-t} + O(n2^{-2t})$

Czyli błąd bezwzględny wynosi

$$|E_n| = |S_n - \hat{S_n}| = |\sum_{i=1}^{n} x_i \xi_i| \leqslant (2 \cdot 2^{-t} + O(n2^{-2t})) \sum_{i=1}^{n} |x_i| \tag{4}$$

Dopóki $n2^{-t} \leqslant 1$ to stała w tym ograniczeniu jest niezależna od n, więc ogólnie ograniczenie jest lepsze niż w zwykłym sumowaniu przedstawionym w rozdziale **??**. Jednak tak samo jak w zwykłym sumowaniu, błąd względny może być duży, gdy elementy ciągu są różnych znaków (gdy suma modułów jest większa niż moduł sumy).

4. Testy

4.1. Wyniki

W celu dokładniejszego sprawdzenia powyższych metod zostały przeprowadzone testy na 3 sumach:

$$\sum_{k=1}^{10000} k^{-2} \tag{5}$$

$$\sum_{k=1}^{10000} (-1) \cdot k^{-2} + 1 \tag{6}$$

$$\sum_{k=1}^{10000} (-1)^k \cdot k^{-2} \tag{7}$$

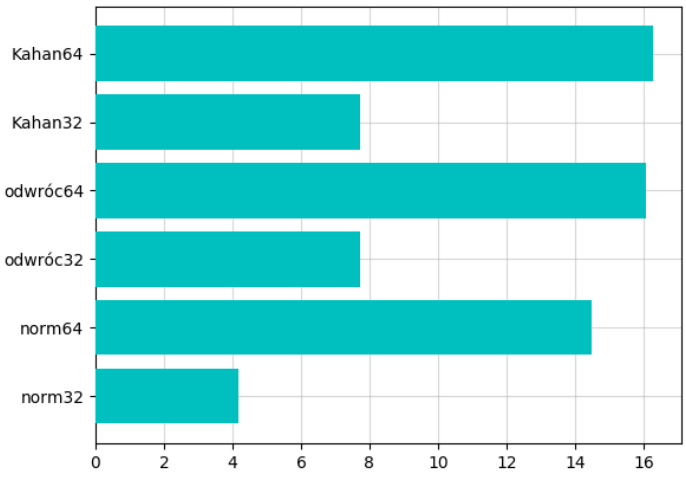
Suma **??** to ciąg odwrotności kwadratów kolejnych liczb naturalnych - w przykładzie z zakresu $[1, 10000]$. Jest to więc suma posortowana malejąco, czyli według wcześniejszych rozważań w złej kolejności. Suma **??** to liczby przeciwne do odwrotności kwadratów kolejnych liczb naturalnych z dodaną liczbą 1 za każdym razem - wszystkie elementy są nieujemne, ale posortowane rosnąco. Ostatnia suma **??** różni się tym od sumy **??**, że jest naprzemienna - jej wyrazy nie są posortowane.

	$\sum_{k=1}^{10000} k^{-2}$	$\sum_{k=1}^{10000}(-1)\cdot k^{-2}+1$	$\sum_{k=1}^{10000}(-1)^k\cdot k^{-2}$
wynik dokładny	1.6448340718480597	9998.355165928151	-0.8224670284246132
suma normalna 32	1.6447253	9998.359	-0.8224671
suma normalna 64	1.6448340718480652	9998.355165928158	-0.8224670284246056
suma odwrócona 32	1.644834	9998.373	-0.822467
suma odwrócona 64	1.6448340718480596	9998.355165928226	-0.8224670284246132
algorytm Kahana 32	1.644834	9998.355	-0.822467
algorytm Kahana 64	1.6448340718480599	9998.355165928151	-0.8224670284246132

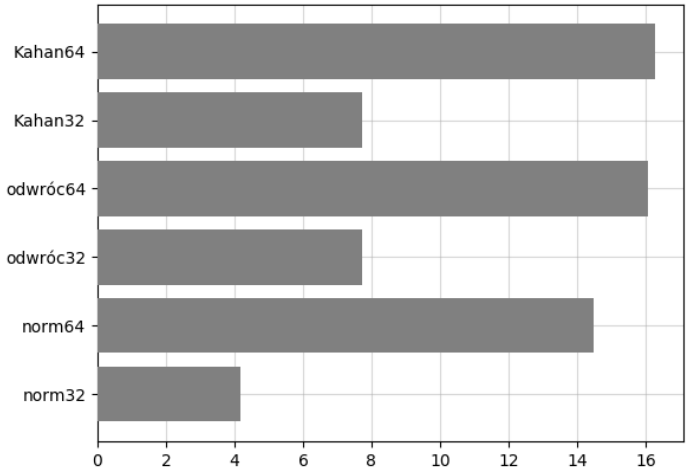
Tabela 1: Wyniki sumowania różnymi algorytmami w różnych precyzjach

Za wyniki dokładne uznaję wartość sumy obliczoną algorytmem Kahana, używając typu BigFloat z precyzją ustawioną na 128 bitów. Wyniki dokładne zostały obcięte bez zaokrąglenia.

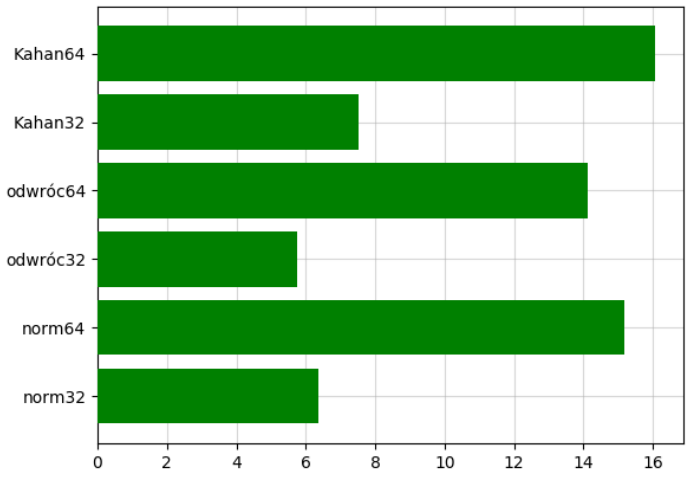
Tabela **??** pokazuje uzyskane wyniki testów w porównaniu do wyników dokładnych. Podkreślone cyfry to cyfry, które zgadzają się z wynikiem dokładnym, co także pokazują rysunki **??**, **??**, **??**. Można zauważyć rozbieżność dla sumy **??** - w tabeli widać, że suma liczona w odwrotnej kolejności (rosnącej) w podwójnej precyzji i algorytm Kahana też w podwójnej precyzji mają tyle samo cyfr dokładnych, natomiast według wykresu wynik policzony algorytmem Kahana ma minimalnie więcej. Związane jest to z zaokrągleniem wartości dokładnej przy liczeniu błędu względnego, przez co wartość dokładna znalazła się bliżej wyniku algorytmu Kahana.



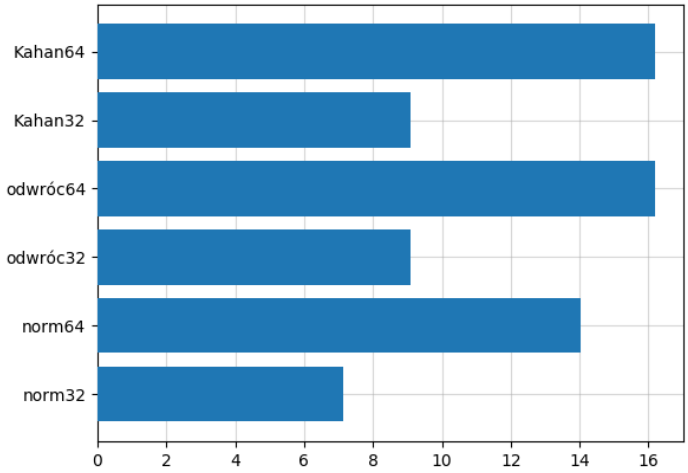
Rysunek 1: Liczba cyfr dokładnych dla sumy **??**



Rysunek 2: Liczba cyfr dokładnych dla sumy elementów przeciwnych jak w sumie **??**



Rysunek 3: Liczba cyfr dokładnych dla sumy **??**



Rysunek 4: Liczba cyfr dokładnych dla sumy **??**

4.2. Analiza wyników

Po wynikach testów widać, że przy zwykłym sumowaniu najlepsze wyniki uzyskujemy, gdy dodajemy liczby posortowane rosnąco. Jednak należy sortować według wartości bezwzględnej liczb, co widać na wykresie **??**, który przedstawia sumę liczb przeciwnych do elementów sumy **??**. Wykresy są identyczne, mimo że elementy były posortowane w sposób odwrotny. Ilustruje to zjawisko opisane w rozdziale **??** - elementy należy sortować co do modułu.

Algorytm Kahana za każdym razem daje tak samo dobre lub lepsze wyniki, niż najlepszy zwykły algorytm sumowania. Patrząc na tabelę (szczególnie na wyniki dla sumy **??**), można zauważyć, że algorytm Kahana w pojedynczej precyzji daje tyle samo cyfr dokładnych, co algorytmy normalnego sumowania w podwójnej przecyzji, gdyby wyniki zaokrąglić do takiej samej liczby cyfr dziesiętnych.

Suma **??** wydaje się nieposortowana, co jest myśleniem błędnym - jest posortowana malejąco co do modułu, więc wyniki zachowują się zgodnie z oczekiwaniami.

5. Podsumowanie

Aby zminimalizować błąd przy sumowaniu liczb zmiennopozycyjnych, można zastosować różne techniki. Najlepszym wyborem jest algorytm Kahana - wykonujemy więcej operacji, ale wynik jest dużo dokładniejszy. Szczególnie jeżeli potrzebujemy wyniku w pojedynczej precyzji.

Jeżeli chcemy ograniczyć liczbę operacji w samym algorytmie dodawania, czyli nie zastosujemy algorytmu Kahana, też mamy możliwość zminimalizować błąd. Trzeba jednak wartości posortować niemalejąco, ale co do modułu. Wynika z tego, że jeśli chcemy policzyć sumę wyrazów ujemnych i dodatnich, należy rozbić sumę na dwie - osobno wyrazów dodatnich i ujemnych (i oczywiście je posortować). Dlatego zwykłe dodawanie opłaca się wykonywać jeżeli mamy już ciąg posortowany (chociaż i tak otrzymujemy potencjalnie gorszy wynik).

Literatura

[1] David Kincaid, Ward Cheney *Analiza Numeryczna*, Warszawa, WNT, 2006.

[2] Germund Dahlquist, Åke Björck *Numerical Methods in Scientific Computing, Vol. I*, SIAM, 2008.

[3] Nicholas J. Higham *The accuracy of floating point summation*, SIAM, 1993.

[4] David Goldberg *What every computer scientist should know about floating-point arithmetic*, Computing Surveys vol.23, 1991.