Maciej Kozub

Laboratorium 1 – Arytmetyka komputerowa

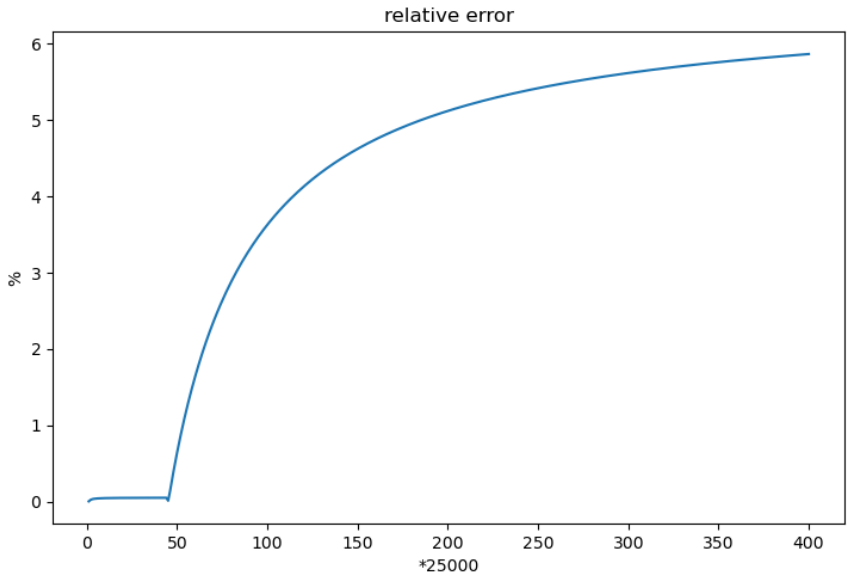
1. Sumowanie liczb pojedynczej precyzji

1.2. Błąd względny wynosi 5,86% i jest on spowodowany błędami zaokrągleń (niedokładnością powstającej stopniowo sumy) i powstaje na całej długości sumowanej tablicy.

1.3.

```
Suma: 2.48253e+06
absolute error: 137527 relative error: 5.86469
```

1.4. Poniższy wykres przedstawia zależność błędu względnego otrzymanego wyniku w zależności od liczby dodanych do siebie wartości (sumowanie iteracyjne).



Początkowo błąd rośnie nieznacznie (dodawane elementy są zbliżonego rzędu wielkości), następnie po ponad 1,25 milionów kroków generowany błąd względny gwałtownie rośnie (niemal logarytmicznie) i finalnie osiąga wartość niemal 6%.

1.5. Wyniki otrzymane przy sumowaniu rekurencyjnym (sumowanie parami) w porównaniu do sumowania iteracyjnego

```
Suma: 2.48253e+06
absolute error: 137527 relative error: 5.86469
Time taken standard ver: 0.139 sec

Suma przy parowaniu: 2.345e+06
absolute error: 0 relative error: 0
Time taken recursive ver: 0.649 sec
```

Jak widać błąd otrzymany przy rekurencyjnym sumowaniu dla N = 10^7 elementów wynosi 0. Błąd ten zmalał z powodu dodawania do siebie liczb o zbliżonej wielkości (ten sam rząd wielkości) z powodu czego nie występują błędy związane z niedokładnością przechowywania cyfr znaczących.

1.6. Porównanie czasów jest dostępne na grafice w poprzednim podpunkcie. Jak widać algorytm rekurencyjny jest niemal 5 razy wolniejszy.

2. Algorytm Kahana

2.1. Dla tych samych danych wejściowych jak w zadaniu 1. Otrzymane błędy względny i bezwzględny wyniosły 0. Wynika z tego że algorytm Kahana jest dokładny.

```
Suma Kahan: 2.345e+06
absolute error: 0 relative error: 0
```

2.2. Algorytm Kahana jest znacznie lepszy ponieważ w przeciwieństwie do tradycyjnego iteracyjnego dodawania nie traci niskich bitów pośrednich sum. Zmienna *err* służy właśnie do przechowywania niskich bitów i jest wykorzystywana do korygowania każdej sumy pośredniej o nie.

2.3. Algorytm rekurencyjny ponownie okazał się być wolniejszy, tym razem około 3,5 razy.

```
Suma przy parowaniu: 2.345e+06
absolute error: 0 relative error: 0
Time taken recursive ver: 0.649 sec

Suma Kahan: 2.345e+06
absolute error: 0 relative error: 0
Time taken Kahan ver: 0.184 sec
```

3. Suma szeregu

3.1. Porównanie wyników dla sumowania w przód i w tył dla pojedynczej precyzji.

```
Suma w przod(50): 0.5 suma w tyl: 0.5
Suma w przod(100): 0.5 suma w tyl: 0.5
Suma w przod(200): 0.5 suma w tyl: 0
Suma w przod(500): 0.5 suma w tyl: 0
Suma w przod(800): 0.5 suma w tyl: 0
```

3.2. Sumowanie dla podwójnej precyzji

```
Suma w przod (double)(50): 0.49999999999999955591079014993738 suma w tyl (double): 0.49999999999999955591079014993738
Suma w przod (double)(100): 0.5 suma w tyl (double): 0.5
Suma w przod (double)(200): 0.5 suma w tyl (double): 0.5
Suma w przod (double)(500): 0.5 suma w tyl (double): 0.5
Suma w przod (double)(800): 0.5 suma w tyl (double): 0.5
```

Dla danego szeregu wartość dokładniejszą niż 0.5 otrzymujemy tylko przy sumowaniu n = 50 elementów.

3.3. Porównując wyniki otrzymane w dwóch poprzednich podpunktach można zaważyć że zaokrąglenie wyniku następuje szybciej przy mniejszej mantysie (zapisanej na mniejszej liczbie bitów).

3.4. Wykorzystanie algorytmu Kahana nie przyniosło żadnych korzyści w obrębie dokładności wyniku w sumowaniu danego szeregu. Wyniki zebrano poniżej.

```
Suma kahan(50): 0.5
Suma kahan(100): 0.5
Suma kahan(200): 0.5
Suma kahan(500): 0.5
Suma kahan(800): 0.5
```

4. Epsilon maszynowy

Epsilon maszynowy to najmniejsza wartość ϵ (najmniejsza nieujemna liczba), której dodanie do 1 da w wyniku 1:

$$1 + \epsilon = 1$$

Aby je wyznaczyć należy jedność dzielić przed podstawę używanego systemu liczbowego do momentu spełnienia powyższej równości. Otrzymany wynik dla podwójnej precyzji:

```
epsilon: 1.11022e-16
```

Jest to liczba, która dodana do jedynki jest nierozróżnialna z jedynką. Ponadto jest zgodna ze standardem IEEE 754.

5. Algorytm niestabilny numerycznie

Niestabilność numeryczna występuje gdy małe błędy danych lub popełniane podczas obliczeń szybko rosną w miarę wykonywania operacji i powodują znaczną niedokładność wyniku. Jako przykład pokażę algorytm obliczania całki oznaczonej rekurencyjnie.

$$y_n = \int_0^1 \frac{x^n}{x+3}dx = \int_0^1 x^{n-1} - \frac{3x^{n-1}}{x+3}dx = \frac{1}{n} - 3y_{n-1}$$

Posługując się tym wzorem rekurencyjnym obliczmy całki dla kolejnych n=0,1,2…

$$y_0 = \int_0^1 \frac{1}{x+3}dx = \ln(4) - \ln(3) = 0{,}288 \qquad |\varepsilon| \le 4 * 10^{-4}$$
$$y_1 = 1 - 3 \cdot 0{,}288 = 0{,}136 \qquad |\varepsilon| \le 12 * 10^{-4}$$
$$y_2 = \frac{1}{2} - 3 \cdot 0{,}136 = 0{,}092 \qquad |\varepsilon| \le 36 * 10^{-4}$$
$$y_3 = \frac{1}{3} - 3 \cdot 0{,}288 = 0{,}057 \qquad |\varepsilon| \le 108 * 10^{-4}$$
$$y_4 = \frac{1}{4} - 3 \cdot 0{,}288 = 0{,}078 \qquad |\varepsilon| \le 324 * 10^{-4}$$
$$y_5 = \tfrac{1}{5} - 3 \cdot 0{,}288 = -0{,}034 \text{ !!!} \qquad |\varepsilon| \le 972 * 10^{-4}$$

Całka oznaczona reprezentuje pole pod wykresem funkcji a więc musi mieć wartość nieujemną, widzimy w tym punkcie że nasz wynik jest bardzo niedokładny. Każdy następny wyraz ciągu może potencjalnie zwiększyć niedokładność 3 razy. Jest to więc algorytm niestabilny numerycznie.

Okazuje się jednak, że można wprowadzić taką modyfikację do algorytmu aby był on stabilny numerycznie. Przekształćmy nasz wzór tak aby móc obliczać poprzedni wyraz mając następny oraz załóżmy np. że wyraz 10 jest równy 9. Mamy wtedy:

$$\frac{1}{3n} - \frac{1}{3}y_n = y_{n-1}, \;\; y_9 = y_{10}$$
$$y_9 = \frac{1}{40} = 0{,}025$$

Zatem obliczmy poprzednie wartości.

$$y_8 = \frac{1}{27} - \frac{1}{3} \cdot 0{,}025 = 0{,}029$$
$$y_7 = \frac{1}{24} - \frac{1}{3} \cdot 0{,}029 = 0{,}032$$
$$y_6 = \frac{1}{21} - \frac{1}{3} \cdot 0{,}032 = 0{,}037$$
$$y_5 = \frac{1}{18} - \frac{1}{3} \cdot 0{,}037 = 0{,}035$$
$$y_4 = \frac{1}{15} - \frac{1}{3} \cdot 0{,}035 = 0{,}055$$
$$y_3 = \frac{1}{12} - \frac{1}{3} \cdot 0{,}055 = 0{,}065$$
$$y_2 = \frac{1}{9} - \frac{1}{3} \cdot 0{,}065 = 0{,}089$$
$$y_1 = \frac{1}{6} - \frac{1}{3} \cdot 0{,}089 = 0{,}137$$
$$y_0 = \frac{1}{3} - \frac{1}{3} \cdot 0{,}137 = 0{,}288$$

Wartość y_0 okazuje się być prawidłowa pomimo potencjalnie dużego błędu początkowego wynikającego z założeń o równości dwóch wyrazów jednak błąd ten nie był propagowany podczas kolejnych obliczeń. Wynika to z faktu że błąd był w każdym kroku dzielony przez 3.