Metody obliczeniowe w nauce i technice
Laboratorium 1
Arytmetyka komputerowa

Adam Dyda

Marzec 2021

1 Wstep

Na laboratorium zajęlismy się sumowaniem dużych ilości liczb zmiennoprzecinkowych, omówieniem błędów które występują w czasie tego procesu oraz algorytmami do radzenia sobie z tymi błędami. Warto zaznaczyć że wszystkie obliczenia wykonywane były w języku C++ z wykorzystaniem typów float i double.

2 Sumowanie liczb pojedynczej precyzji

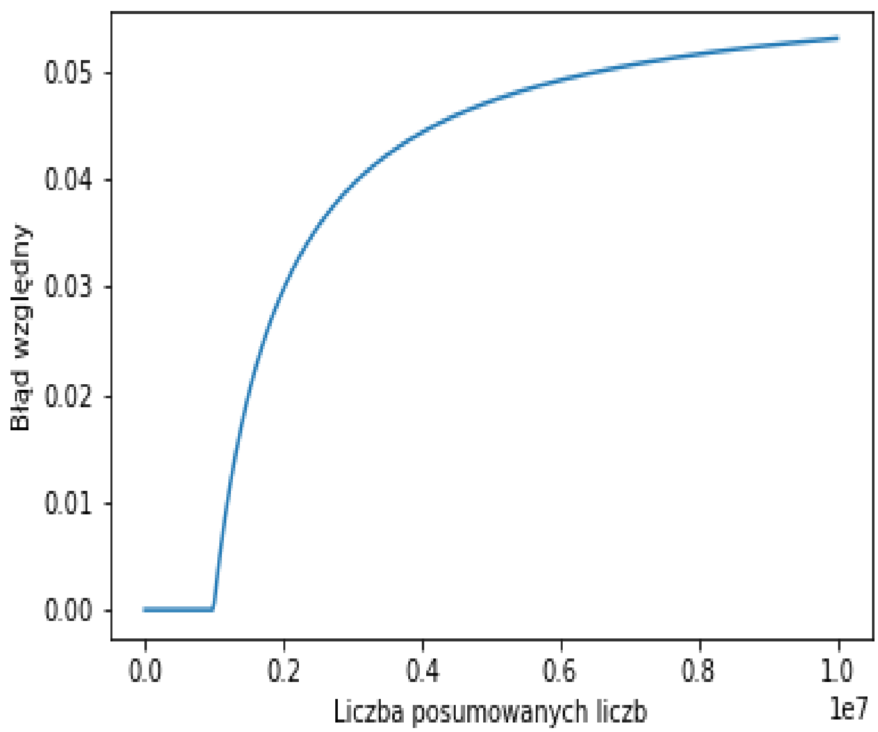
2.1 Sumowanie N liczb pojedynczej precyzji przechowywanych w tablicy

Dla danej tablicy o N = 10^7 elementach, wypełnioną wartościa $v = 0.53125$ wynik sumowania takiej tablicy działając na liczbach zmiennoprzecinkowych pojedynczej precyzji wynosi 5030840.5 podczas gdy realna wartość powinna wynosić 5312500. Wyznaczmy teraz błąd bezwzględny i względny obliczeń:

Błąd bezwględny wynosi: 281659.5
Błąd względny wynosi: 0.05301825702

Możemy zaobserwować że zarówno błąd względny i bezwzględny jest duży, wynika to z prostego algorytmu sumowania w którym błąd przy sumowaniu liczb zmiennoprzecinkowych kumuluje się z kolejnymi operacjami.

Aby zobaczyc jak rośnie błąd względny w trakcie sumowania. Przedstawimy wykres błedu co 25000 kroków. Na wykresie możemy zauważyć że dla początkowych wartości bład jest bliski zeru. Ze wzrostem liczby operacji bład rośnie jest to spowodowane wczesniej już wspomnianym kumulowaniem się błedu z kolejnymi krokami.



2.2 Algorytm rekurencyjny sumowania

Dla tych samych danych wejsciowych spróbujemy zsumować tablice wykorzystując algorytm rekurencyjny. W tym wypadku wynik sumowania wynosi 5312500 czyli tyle samo co wartość realna. Stąd oczywiście błąd wględny i bezwzględny bedzie wynosić zero. Wynika to z tego że w tym przypadku nasz błąd nie kumuluje się jak wcześniej, liczby które dodajemy do siebie w kolejnych krokach są do siebie zbliżone wielkościa ponieważ nie akumulujemy wartosci sumy tylko w jednej zmiennej.

Aby otrzymać niezerowy błąd należy wybrać odpowiednie dane wejsciowe, dane dla których udało mi sie otrzymać niezerowy błąd to talica o N = 10^7 elementach wypełnioną wartościa 0.333333. Błedy w tym wypadku są nastepujące:

Błąd bezwględny wynosi: 0.25
Błąd względny wynosi: $7.500008063 * 10^{-8}$

3 Algorytm Kahana

Zaimplementowałem algorytm Kahana do sumowania tablicy liczb w celu zmniejszenia błedów związanych z sumowaniem. W przypadku algorytmu Kahana zarówno błąd względny jak i bezwzględny wynoszą zero. Jest to spowodowane tym że algorytm Kahana kompensuje błędy zapomocą zmiennej err w której przechowywane są błędy kolejnych operacji zmiennoprzecinkowych.

4 Porównanie czasu działania algorytmów

Poniżej przedstawiam czasy wykonania powyższych algorytmów dla następujących danych wejściowych, tablica o wielkości $N = 10^7$, wypełniona wartością $v = 0.53125$.

Prosty algorytm sumowania: 3059 mikrosekund
Rekurencyjny algorytm sumowania: 83843 mikrosekund
Algorytm Kahana: 95516 mikrosekund

Jak widać skomplikowanie algorytmu oczywiście wiąże się z kosztem na czasie jego wykonania.

5 Sumy częściowe

Nastepnie zająłem sie obliczaniem sum częściowych szeregu definującego funkcję dzeta Riemanna oraz funkcję eta Dirichleta. Obliczyłem sumy dla różnych wartości s = 2, 3.6667, 5, 7.2, 10 oraz n = 50, 100, 200, 500, 1000 sumując w przód jak i wstecz, używając, liczb pojedynczej i podwójnej prezycji.
Patrząc na wyniki można zauważyć że w przypadku liczb pojedynczej precyzji wyniki sumując w przód i sumując wstecz różnią się. Wynika to z tego że dodawanie liczb zmiennoprzecinkowych nie jest przemienne. Dodając wstecz dochodząc do poczatku naszego ciągu dodajemy duże liczby do małych liczb natomiast sumując w przód na końcu naszego ciagu dodajemy duże liczby do dużych liczb, co może mieć inny rezultat jeżeli chodzi o generowane błędy. Warto zauważyć że w przypadku liczb podwójnej prezycji dla powyższych ciągów i danych różnice w sumach nie występowały.

Rysunek 1: Sumy częściowe szeregu definiującego funkcję dzeta Riemanna, dla liczb pojedynczej precyzji.

s\N	50		100		200		500		1000	
	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz
2	1.625132918	1.625132799	1.634984016	1.634983897	1.639946699	1.639946461	1.642935991	1.642935991	1.643934846	1.643934488
3.6667	1.109399438	1.109399796	1.109408617	1.109408855	1.109408617	1.109410286	1.109408617	1.109410524	1.109408617	1.109410524
5	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277
7.2	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659
10	1.000994563	1.000994563	1.000994563	1.007227659	1.000994563	1.000994563	1.000994563	1.000994563	1.000994563	1.000994563

Rysunek 2: Sumy częściowe szeregu definiującego funkcję eta Dirichleta, dla liczb pojedynczej precyzji.

s\N	50		100		200		500		1000	
	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz	W przód	Wstecz
2	1.625132918	1.625132799	1.634984016	1.634983897	1.639946699	1.639946461	1.642935991	1.642935991	1.643934846	1.643934488
3.6667	1.109399438	1.109399796	1.109408617	1.109408855	1.109408617	1.109410286	1.109408617	1.109410524	1.109408617	1.109410524
5	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277	1.036927462	1.0369277
7.2	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659	1.007227659
10	1.000994563	1.000994563	1.000994563	1.007227659	1.000994563	1.000994563	1.000994563	1.000994563	1.000994563	1.000994563