Metody Obliczeniowe w Nauce i Technice
Laboratorium 1
Arytmetyka komputerowa

Piotr Stecyk

Wstęp

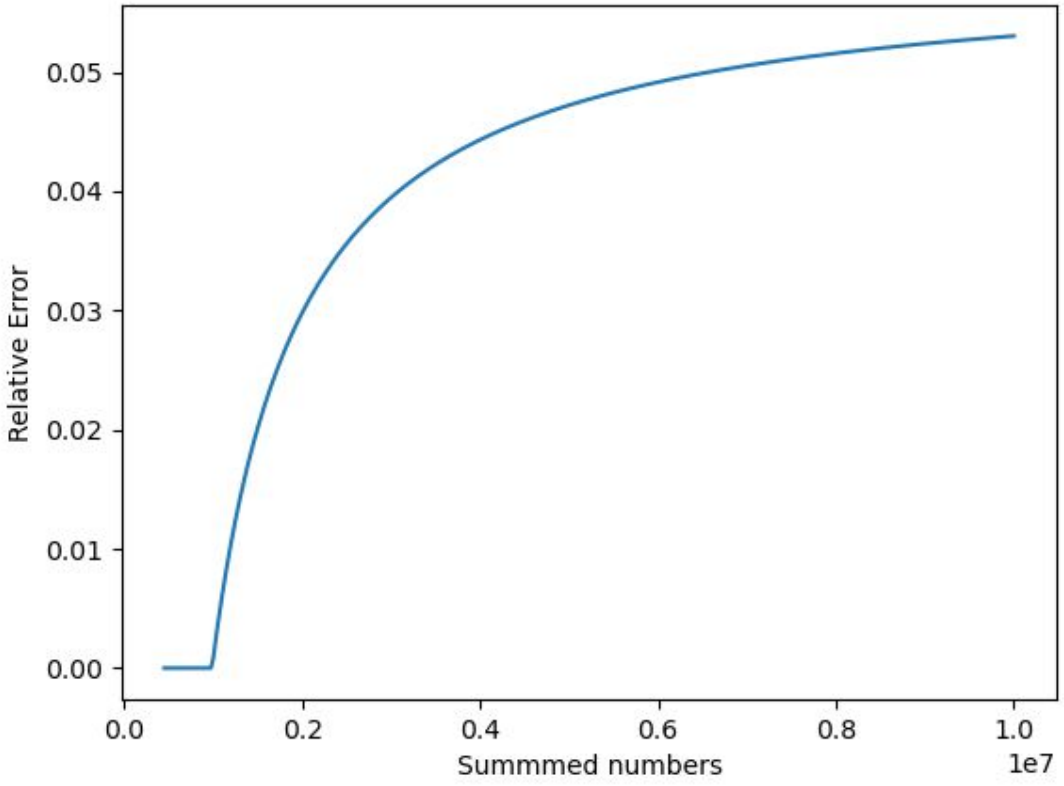
Na pierwszym laboratorium przyjrzeliśmy się arytmetyce komputerowej oraz wpływu niedokładności reprezentacji liczb zmiennoprzecinkowych na wyniki różnych działań.

Zad 1

W pierwszym zadaniu skupiliśmy się na problemie związanym z sumowaniem liczb zmiennoprzecinkowych pojedynczej precyzji. Pierwsza metoda - klasyczne dodawanie po kolei liczb - okazało się bardzo niedokładna. Dla tablicy o rozmiarze n = 10^7 wypełnionej tą samą wartością v = 0.53125. Błąd względny i bezwzględny okazał się bardzo duży.

Błąd bezwzględny: 281659.5
Błąd względny: 0.0530183

Jest to spowodowane tym, że podczas dodawania po kolei liczb zmiennoprzecinkowych nasz błąd się kumuluje, im więcej cyfr, tym większy błąd otrzymujemy. Poniższy wykres przedstawia w jaki sposób rośnie błąd względny, pomiary robione były co 25000 kroków.



Jak widzimy na początku błąd był bliski 0. Dopiero od pewnego momentu zaczął gwałtownie rosnąć i się kumulować.
Następnie sprawdziliśmy działanie rekurencyjnego algorytmu sumowania. Okazało się, że ta metoda jest poprawniejsza. Błąd względny i bezwzględny przy tych samych danych wejściowych wyszedł 0. Błąd zmalał, ponieważ w tym przypadku nasz błąd nie kumuluje się tylko w jednej zmiennej czyli w sumie, tylko rozdzielamy to sumowanie rekurencyjnie.
Kosztem mniejszych błędów jest czas wykonania takiego algorytmu.

Dla tych samych danych wejściowych, algorytmu uzyskały następujące czasy:
standardowe sumowanie: 25495ms
rekurencyjne sumowanie: 57884ms

Jak widzimy rekurencyjne sumowanie jest dwa razy wolniejsze, aczkolwiek stosując je możemy mieć pewność dokładniejszego wyniku.

Testując program, udało mi się uzyskać niezerowy błąd dla wartości równej v = 0.666666

Zad 2.

W zadaniu tym zaimplementowaliśmy algorytm Kahana do sumowania liczb. Dla tych samych danych wejściowych błąd względny i bezwzględny wyniosły 0. Algorytm ten ma znacznie lepsza własności numeryczne, ponieważ zmienna *err* na bieżąco pozwala nam wprowadzać poprawki naszego sumowania. Zmienna *err* przechowuje naszą poprawkę związaną z dodawaniem liczby do sumy końcowej, z każdą iteracją poprawka jest aktualizowana i odpowiednio wprowadzana do naszej sumy końcowej.
Jako że musimy w tym rozwiązaniu wykonać więcej obliczeń, czas wykonania tego algorytmu jest najdłuższy, aczkolwiek ma on najlepszą precyzję.

Czasy:
standardowe sumowanie: 25495ms
rekurencyjne sumowanie: 57884ms
algorytm Kahana: 79810ms

Zad 3

W trzecim zadaniu zaimplementowaliśmy dwie funkcję, dzeta Riemanna oraz eta Dirichleta. Oto wyniki dla różnych danych wejściowych otrzymane przez funkcje Riemanna.

Results for float precision.										
n	**50**		**100**		**200**		**500**		**1000**	
s	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD
2	1.625132918	1.625132799	1.6349840164	1.6349838972	1.6399466991	1.6399464607	1.6429359913	1.6429359913	1.6439348459	1.6439344883
3.6667	1.109399437	1.109399795	1.109408617	1.1094088554	1.109408617	1.1094102859	1.109408617	1.1094105244	1.109408617	1.1094105244
5	1.036927461	1.0369277	1.0369274616	1.0369277	1.0369274616	1.0369277	1.0369274616	1.0369277	1.0369274616	1.0369277
7.2	1.007227659	1.007227659	1.0072276592	1.0072276592	1.0072276592	1.0072276592	1.0072276592	1.0072276592	1.0072276592	1.0072276592
10	1.000994563	1.000994563	1.0009945631	1.0009945631	1.0009945631	1.0009945631	1.0009945631	1.0009945631	1.0009945631	1.0009945631

Results for double precision.										
n	**50**		**100**		**200**		**500**		**1000**	
s	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD
2	1.6251327336	1.6251327336	1.6349839002	1.634983900	1.639946546	1.639946546	1.642936065	1.6429360655	1.643934566	1.6439345667
3.6667	1.109399766	1.109399766	1.1094088082	1.109408808	1.109410253	1.109410253	1.109410501	1.1094105017	1.109410521	1.1094105217
5	1.0369277167	1.0369277167	1.0369277527	1.036927752	1.036927755	1.036927755	1.036927755	1.0369277551	1.036927755	1.0369277551
7.2	1.0072276675	1.0072276675	1.0072276675	1.007227667	1.007227667	1.007227667	1.007227667	1.0072276675	1.007227667	1.0072276675
10	1.0009945751	1.0009945751	1.0009945751	1.000994575	1.000994575	1.000994575	1.000994575	1.0009945751	1.000994575	1.0009945751

oraz dla funkcji eta Dirichleta

Results for float precision.										
n	**50**		**100**		**200**		**500**		**1000**	
s	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD
2	0.82227098942	0.8222710490	0.8224174976	0.8224174976	0.822454690	0.822454571	0.822465360	0.8224650621	0.822466850	0.82246649265
3.6667	0.93469303846	0.9346930384	0.9346932172	0.9346933364	0.934693217	0.934693336	0.934693217	0.9346933364	0.934693217	0.93469333649
5	0.9721198082	0.9721197485	0.9721198082	0.9721197485	0.972119808	0.972119748	0.972119808	0.9721197485	0.9721198082	0.97211974859
7.2	0.99352705479	0.9935269951	0.9935270547	0.9935269951	0.993527054	0.993526995	0.993527054	0.9935269951	0.993527054	0.99352699518
10	0.99903953075	0.9990395307	0.9990395307	0.9990395307	0.999039530	0.999039530	0.999039530	0.9990395307	0.999039530	0.99903953075

Results for double precision.										
n	50		100		200		500		1000	
s	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD	FORWARD	BACKWARD
2	0.82227104902	0.8222710490	0.82241755724	0.822417557	0.822454571	0.822454571	0.822465062	0.8224650621	0.822466552	0.82246655226
3.6667	0.93469303846	0.9346930384	0.93469333649	0.934693336	0.934693336	0.934693336	0.934693336	0.9346933364	0.934693336	0.93469333649
5	0.97211974859	0.9721197485	0.97211974859	0.972119748	0.972119748	0.972119748	0.972119748	0.9721197485	0.9721197485	0.97211974859
7.2	0.99352699518	0.9935269951	0.99352699518	0.993526995	0.993526995	0.993526995	0.993526995	0.9935269951	0.993526995	0.99352699518
10	0.99903953075	0.9990395307	0.99903953075	0.999039530	0.999039530	0.999039530	0.999039530	0.9990395307	0.999039530	0.99903953075

Jak widzimy, wyniki różnią się od siebie. Dla s = 2 tablicowa wartość dzeta Riemanna wynosi około 1.6449341. Im większe n tym wartość naszej funkcji jest zbliżona do wartości oczekiwanej. Jak widzimy wyniki dla funkcji zliczającej w przód oraz funkcji zliczającej wstecz różnią się. Dzieje się tak dlatego, że kolejność sumowania liczb zmiennoprzecinkowych ma znaczenie, ponieważ w trakcie działania kumuluję się inny błąd, aczkolwiek w tym przypadku różnice są naprawdę marginalne. Tak samo marginalne wyszły mi różnice między zastosowaniem liczb pojedynczej prezycji oraz podwójnej precyzji.