Algorytmy numeryczne

Zadanie 1
Aleksander Kosma
grupa 1 tester-programista

16 Październik 2017

1 Sumowanie szeregów potęgowych

Opracowanie prezentuje próby obliczenia funkcji arctan, przy pomocy języka java. Wykorzystano w tym celu dwa sposoby na obliczenie tej wartości. Są to:
-zsumowanie elementów wyliczonych z szeregu potęgowego:

$$arctg(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} x^{2n+1}$$

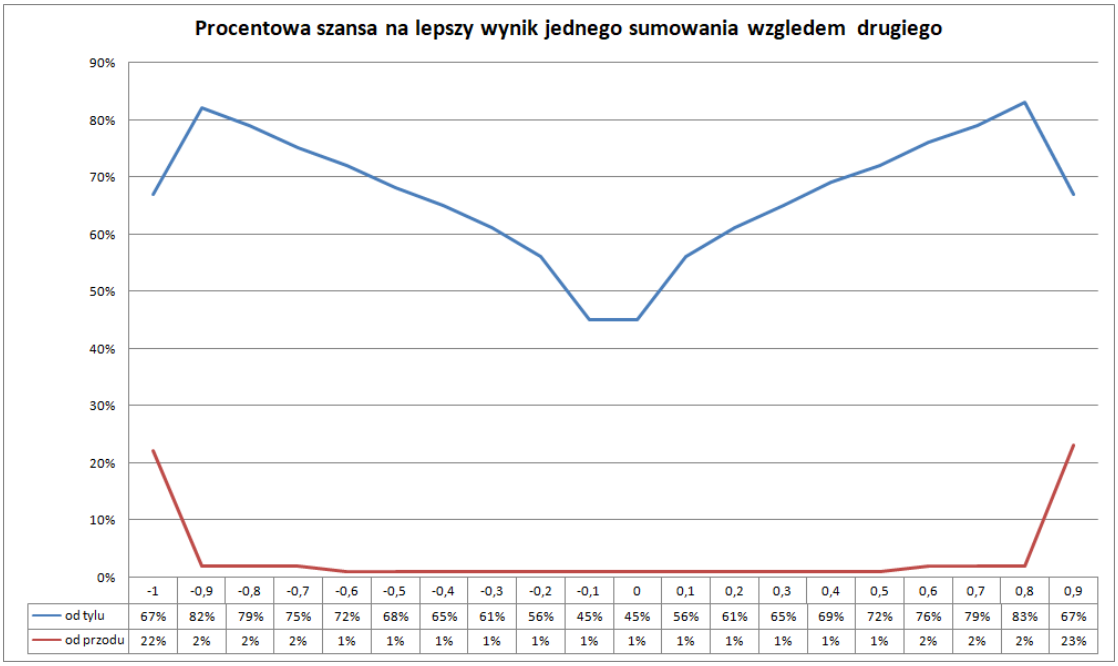
-zsumowanie elementów wyliczonych na podstawie poprzednika:

$$a_{n+1} = a * -\frac{x^2 * (2n+1)}{2n+3}$$

Ze względu na charakter obliczeń podzieliłem owe zsumowania na kolejne dwa:
-zsumowanie elementów licząc od tyłu
-zsumowanie elementów licząc od przodu

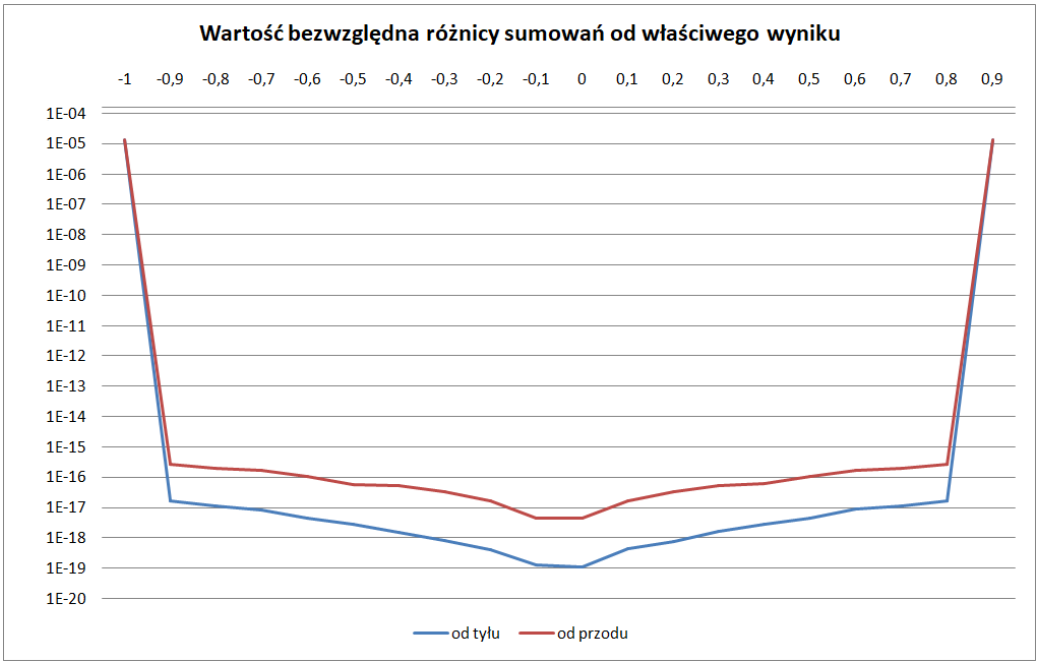
Wnioski zostały obliczone na zmiennej typu double. Również ze względu na zbieżność szeregu w $[-1, 1]$ liczyłem tylko dla tego zakresu.

Wykres przedstawia który rodzaj sumowania ma większą szanse na precyzyjniejszy wynik względem drugiego. Każda kolumna wartości została wyliczona dla 100 tysięcy liczb między podanymi przedziałami. Razem 2 miliony próbek.

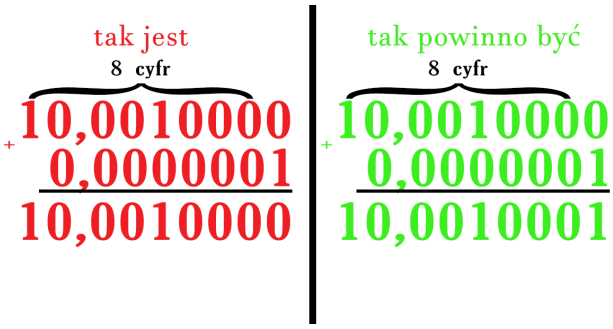


	-1	-0,9	-0,8	-0,7	-0,6	-0,5	-0,4	-0,3	-0,2	-0,1	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
od tylu	67%	82%	79%	75%	72%	68%	65%	61%	56%	45%	45%	56%	61%	65%	69%	72%	76%	79%	83%	67%
od przodu	22%	2%	2%	2%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	1%	2%	2%	2%	23%

Drugi wykres prezentuje bezwzględną różnice miedzy wynikiem biblioteki a sumowaniami. Środkowa część wykresu ukazuje wyższość sumowania od tyłu. Jest ono precyzyjniejsze nawet o 2 rzędy wielkości.



Takie rezultaty wynikają z natury działania zmiennej double. W przypadku sumowania liczby większej do liczby mniejszej(gdzie różnica wynosi parę rzędów wielkości), zmienna odrzuca końcowe cyfry mniejszej liczby by pomieścić najistotniejsze wartości. Stąd utrata precyzji. W momencie kiedy dwie liczby mają podobną wartość, ich suma potrzebować będzie podobnej precyzji do zapisania wyniku. W konsekwencji nie odrzuci danych. Podsumowując kwestie precyzji wyniku. Opierając się o zmienną double, można stwierdzić, że precyzja wyniku waha się między 14, a 16 miejscem po przecinku.
Na obrazku poniżej znajduje się praktyczny przykład. Dla uproszczenia zagadnienia jest on prezentowany na zmiennej float(32 bity) która zazwyczaj mieści w sobie 8 cyfr dziesiętnych.



By rozwiązać problem niskiej precyzji, który zaistniał w przypadku doubla, trzeba mieć kontrolę nad precyzją obliczeń. Rozwiązaniem może okazać się klasa BigDecimal. Klasa ta może przechować nieograniczoną wartość liczbową, ogranicza ją jedynie skala która osiąga wartość 32-bitowego integera, czyli trochę ponad dwa miliardy miejsc.
Udało mi się osiągnąć praktycznie nieograniczoną precyzję. Wyniki oparte o BigDecimal porównywałem z kalkulatorem online Wolfram(computational knowledge engine). Przykładowy wypis programu wraz ze wszystkimi rodzajami obliczeń i ich wynikami. W tym przypadku podana precyzja BigDecimal i Wolfram to 64 miejsca po przecinku.

```
Output of Math libary:              0.2303949468807241
Output of Wolfram:                  0.230394946880724096035195401834049262605406955181038715333177664 0

Sum forward McLourin BigDecimal:    0.230394946880724096035195401834049262605406955181038715333177664 0
Sum backward McLourin BigDecimal:   0.230394946880724096035195401834049262605406955181038715333177664 0

Sum forward McLourin Double:        0.23039494688072407
Sum backward McLourin Double:       0.2303949468807241

Sum forward next BigDecimal:        0.230394946880724096035195401834049262605406955181038715333177664 0
Sum backward next BigDecimal:       0.230394946880724096035195401834049262605406955181038715333177664 0

Sum forward next Double:            0.23039494688072407
Sum backward next Double:           0.2303949468807241
```