# Implementacja rozwinięć wybranych funkcji w szereg Maclaurina

Grams, Stanisław

17 października 2018

# 1 Sumowanie szeregów potęgowych

## 1.1 Rozwinięcia funkcji w szereg Maclaurina

$$\forall x \sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$$

$$\forall x \exp x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

## 1.2 O implementacji

Program "taylor" został napisany w języku C z użyciem bibliotek "quadmath.h", "pthread.h" oraz "glib.h" pozwalających na użycie 128-bitowych zmiennych typu zmiennoprzecinkowego, wielowątkowości oraz ułatwiających pracę z tablicami znaków.
Wyniki działania są zapisywane do poszczególnych plików **.csv*.

## 1.3 Wyniki

Program uruchamiano na przedziale $x \in [-10; 9.99999]$ dla kolejnych parametrów $M = 4, 8, 16, 32, 64$. Z powodu braku wspomagania koprocesora zmiennoprzecinkowego dla typu poczwórnej precyzji obliczenia zajmowały dużo czasu (dla M=256 nawet 16 godzin!).

## 1.4 Konkluzje po implementacji i przeprowadzeniu doświadczeń

### 1.4.1 Odpowiedzi na postawione hipotezy

**Hipoteza 1.** Sumowanie od końca daje dokładniejsze wyniki niż sumowanie od początku.

Hipoteza, dla zaimplementowanych funkcji, na przedziale $x \in [-10; 9.99999]$, jest prawdziwa **jedynie** w pierwszym przypadku, tj. sumując elementy szeregu potęgowego obliczane bezpośrednio ze wzoru Taylora w kolejności od końca.

**Hipoteza 2.** Używając rozwinięcia wokół 0 (szereg Maclaurina), przy tej samej liczbie składników szeregu dokładniejsze wyniki uzyskujemy przy małych argumentach.

Zaimplementowane funkcje pokazują, że na przedziale $x \in [-10; 9.99999]$ hipoteza jest prawdziwa, co możemy łatwo zauważyć na wykresie "01_functions_m002", gdzie funkcje 1,2,3 oraz 4 zbiegają się z wykresem funkcji 0 tylko w okolicach zera oraz w całkowitych wielokrotnościach liczby $2\pi$ (ze względu na użyte sprowadzenie do przedziału $[-2\pi; 2\pi]$.

**Hipoteza 3.** Sumowanie elementów obliczanych na podstawie poprzedniego daje dokładniejsze wyniki niż obliczanych bezpośrednio ze wzoru.

Zaimplementowane funkcje pokazują, że na przedziale $x \in [-10; 9.99999]$ hipoteza jest prawdziwa, co więcej, możemy powiedzieć, że w przypadku użycia typu __float128, funkcje te mają podobną dokładność co funkcja 2 (szereg Maclaurina, sumowanie od końca). Dowodem niech będzie wynik średnich różnic w stosunku do obliczeń popełnionych funkcjami wbudowanymi:

```
func1() avg. absolute diff to func0() with M=16 is
0.00000000000165395217963605979859070783031087967536128775075031 84
func2() avg. absolute diff to func0() with M=16 is
0.00000000000165395217963605979859082800559813574406369386428509 51
func3() avg. absolute diff to func0() with M=16 is
0.00000000000165395217963605979859082800559813574406369386428509 51
func4() avg. absolute diff to func0() with M=16 is
0.00000000000165395217963605979859082800559813574406369386428509 51
```

Na powyższych wynikach, dla parametru M=16, zauważyć możemy dokładność co do 33 miejsc po przecinku.

### 1.4.2 Odpowiedzi na postawione pytania

**Pytanie 1.** Jak zależy dokładność obliczeń (błąd) od liczby sumowanych składników?

Im wyższa liczba sumowanych składników (parametr M), tym większa dokładność obliczeń (niższy błąd bezwzględny do obliczeń wykonanych przez funkcje wbudowane), oraz tym dłuższy czas wykonania programu.

## 1.5 Spostrzeżenia

1. Użycie __float128 - typu bez wspomagania wbudowanego koprocesora znacznie zwiększa czas działania programu.

2. Spośród zbadanych parametrów wystarczająca dokładność uzyskiwana jest już od M=32.

3. Sprowadzanie argumentu do przedziału $[-2\pi; 2\pi]$ zwiększa dokładność obliczeń oraz przyspiesza ich czas.

4. Znaczący przyrost wydajności uzyskać można przez podzielenie obliczeń pomiędzy kilka wątków.

## 1.6 Wykresy

Wykresy oraz wyniki średnich arytmetycznych załączono w katalogu *report/plots/*. Użyto oprogramowania *Octave* wraz z pakietem *gnuplot*. Skrypty załączono w katalogu *scripts/*.