Zadanie N01 - Sprawozdanie

Jakub Dziurka

Program jest napisany w Pythonie i ma na celu obliczenie wartości funkcji matematycznej na podstawie zadanej sumy nieskończonej. Każdy wyraz szeregu jest obliczany zgodnie z formułą:

$$term = sin(nx^4)^2 e^{-n} + cos(nx^4)^2 e^{-4n}$$

gdzie x jest wartością wejściową do funkcji $f(x, N)$.

Całkowity wynik obliczeń to suma wartości term dla każdego n w zakresie od 0 do 19. Każdy wyraz serii składa się z dwóch części: kwadratu funkcji sinus pomnożonego przez eksponencjalnie malejący czynnik oraz kwadratu funkcji cosinus pomnożonego przez szybciej eksponencjalnie malejący czynnik.

Z uwagi na kwadratowe potęgi funkcji sinus i cosinus, każdy wyraz serii jest zawsze nieujemny. Funkcje eksponencjalne e^{-n} i e^{-4n} powodują, że wkład poszczególnych wyrazów do sumy serii maleje w miarę wzrostu n.

(a) wynik obliczeń dla f (1) z dokładnością do 10 cyfr znaczących:

Wynik: 1.396442121

(b) ilość obliczeń zdefiniowaną jako sumę:

Liczba N: 20

Liczba operacji: 1529.5

Uzasadnienie wyboru N:

Liczbę N przyjąłem jako 20. W miarę jak n rośnie, dodatkowe wyrazy w szeregu dają coraz mniejszy wkład do końcowego wyniku. Dodanie kolejnych wyrazów nie wpływa znacząco na wynik w ramach wymaganej precyzji (10 cyfr znaczących), więc osiągnięto odpowiednią dokładność i dalsze iteracje nie są potrzebne. Z tego względu wystarczy przejść przez pętle 20 razy.

Użyte optymalizacje:

1. **Unikanie Powtarzających Się Obliczeń:**
   - Wartość x_pow = x ** 4 jest obliczana tylko raz przed pętlą. To zapobiega wielokrotnemu obliczaniu tej samej wartości w każdej iteracji pętli.

2. **Optymalizacja Obliczeń Trygonometrycznych:**
    - Wykorzystanie wzorów na sumę kątów dla sinusa i cosinusa umożliwia efektywniejsze obliczanie wartości sin(n * x_pow) i cos(n * x_pow) bez konieczności wielokrotnego wywoływania funkcji math.sin i math.cos.
3. **Rekurencyjne Obliczanie Eksponenty:**
    - Wartości exp(-n) i exp(-4n) są obliczane rekurencyjnie. Zamiast obliczać te wartości w każdej iteracji pętli, mnożymy poprzednie wartości odpowiednio przez exp_minus_1 i exp_minus_4, co jest obliczeniowo bardziej efektywne.
4. **Optymalizacja Pętli:**
    - Pętla w funkcji f jest wykonywana dokładnie N razy (gdzie N = 20). To ograniczenie liczby iteracji do stałej wartości pozwala na kontrolę nad złożonością obliczeniową algorytmu.
5. **Wykorzystanie Funkcji Wbudowanych:**
    - Korzystamy z funkcji matematycznych dostarczanych przez bibliotekę math, co jest zazwyczaj szybsze niż ręczna implementacja tych funkcji.