Student : Mateusz Rachwał, 269043
Prowadzący : dr inż. Mateusz Żbikowski

Politechnika Warszawska
Wydział Mechaniczny Energetyki i Lotnictwa
Obliczenia inżynierskie w chmurze
Obliczanie całki oznaczonej metodą trapezów

Warszawa, 2022

1 Streszczenie

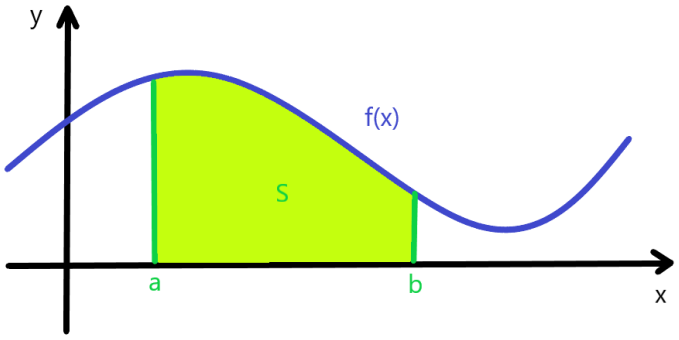
W projekcie poznano platformę Microsoft Azure umożliwiającą dostęp do chmury obliczeniowej. Stworzono tam maszynę wirtualną i na niej uruchomiono program za pomocą którego wykonano szereg obliczeń całek oznaczonych funkcji rzeczywistych jednej zmiennej metodą trapezów. Przeanalizowano wpływ długości kroku całkowania i liczby wątków użytych do obliczeń na czas wykonywania programu.

2 Wstęp teoretyczny

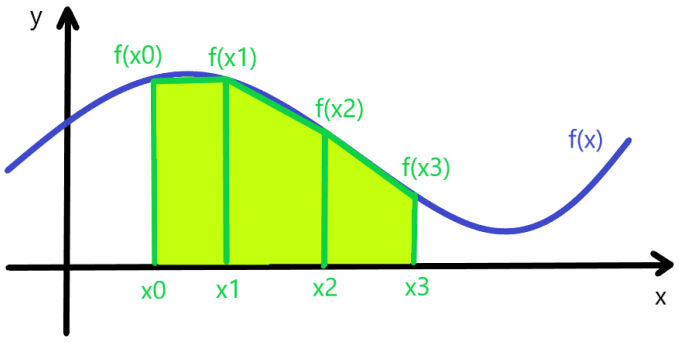
2.1 Informacje ogólne

Na platformie Microsoft Azure utworzono konto, następnie maszynę wirtualną. Maszyna wirtualna z systemem Ubuntu. Z maszyną łączono się za pomocą programu PuTTY, program w Pythonie do obliczeń całek przerzucono na maszynę wirtualną za pomocą programu WinSCP.

2.2 Metoda trapezów



Rysunek 1: Wykres przedstawiający całkę oznaczoną z funkcji $f(x)$ na przedziale od a do b jako pole pod wykresem funkcji w tym przedziale S



Rysunek 2: Wykres przedstawiający obliczanie całki metodą trapezów

Całkę oznaczoną funkcji jednej zmiennej rzeczywistej $f(x)$ w przedziale od a do b możemy traktować jako pole S pod wykresem tej funkcji w tym przedziale zgodnie z Rysunkiem 1. Pole pod tym wykresem możemy przybliżać trapezami - Rysunek 2, mówimy wówczas o metodzie trapezów. Dzielimy przedział całkowania na N równych odcinków zwanych krokiem całkowania $h = x_{i+1} - x_i$, mamy wtedy $N + 1$ punktów (od x_0 do x_N).
Pole pojedynczego trapezu możemy zapisać :

$$S_i = \frac{1}{2}(f(x_i) + f(x_{i+1}))h \tag{2.2.1}$$

Pole całej powierzchni S, czyli wartość całki oznaczonej w danym przedziale to suma pól wszystkich trapezów :

$$\int_a^b f(x)dx = \int_{x_0}^{x_N} f(x)dx = \sum_{i=0}^{N-1} \frac{1}{2} \left(f(x_i) + f(x_{i+1})\right) h \tag{2.2.2}$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_N} f(x)dx = \left(\frac{1}{2}f(x_0) + f(x_1) + f_{x_2} + ... + f(x_{N-2}) + f(x_{N-1}) + \frac{1}{2}f(x_N)\right) h \tag{2.2.3}$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_N} f(x)dx = \left(f(x_0) + f(x_1) + ... + f(x_{N-1}) + f(x_N)\right) h - \frac{1}{2} \left(f(x_0) + f(x_N)\right) h \tag{2.2.4}$$

$$\int_a^b f(x)dx = \int_{x_0}^{x_N} f(x)dx = \left(\sum_{i=0}^{N} f(x_i)h)\right) - \frac{1}{2} \left(f(x_0) + f(x_N)\right) h \tag{2.2.5}$$

Wzór (2.2.5) jest zaimplementowany w programie.

2.3 Kod programu

2.3.1 Obsługa programu

```
matti@Maszyna02:~$ python3 CalkTrapez.py
Enter function that will be integrated: cos(x)
Enter the step: 0.00001
Enter the lower integration bound: 0
Enter the upper integration bound: 1.5708
Enter the number of processes used to computation: 100
0.9994947163579875
```

Rysunek 3: Przykładowe użycie programu, obliczano całkę z funkcji $\cos(x)$ z krokiem całkowania 0.00001, przedział od 0 do 1.5708, utworzone 100 wątków do obliczeń

Na Rysunku 3 przedstawiono przykładowe uruchomienie programu. Użytkownik uruchamia i obsługuje program za pomocą terminalu. Wprowadza całkowaną funkcję, krok całkowania, dolną i górną granicę przedziału całkowania i liczbę wątków utworzonych do obliczeń. Po zakończeniu obliczeń program drukuje wynik do terminalu.

2.3.2 Środowisko

Program liczący całki jest napisany w języku Python. Wykorzystuje on moduły *math* oraz *multiprocessing*.
Moduł *math* składa się z funkcji opakowujących funkcje ze standardowej biblioteki matematycznej języka C.
Moduł *multiprocessing* dostarcza mechanizmów tworzenia procesów wykorzystując API podobnego do tego używanego w module *thread*. Oferuje ona funkcjonalność przetwarzania współbieżnego zarówno lokalnie jak i zdalnie, pozwalając na obchodzenie niektórych blokad dzięki wykorzystaniu podprocesów zamiast oddzielnych wątków. Z uwagi na to, moduł pozwala na pełniejszą kontrolę i wykorzystanie dostępnych zasobów komputera. Moduł jest kompatybilny zarówno z systemami Linux jak i Windows. Moduł *multiprocessing* dodatkowo zawiera funkcjonalności nie występujące w klasycznym module *thread*. Najważniejsza z nich są obiekty typu **Pool**, które dostarczają wygodnych metod obliczeń równoległych, pomagając rozdzielać dostępne zasoby.

2.3.3 Zmienne

foo - zmienna typu string, przechowuje funkcję wprowadzoną przez użytkownika,
skok - zmienna typu float, przechowuje wprowadzoną przez użytkownika wartość kroku całkowania,
a - zmienna typu float, przechowuje wartość dolnej granicy całkowania, wprowadzana przez użytkownika,
b - zmienna typu float, przechouje wartość górnej granicy całkowania, wprowadzana przez uzytkownika,
processes - zmiennna typu int, przechowuje informacje o liczbie wątków tworzonych w czasie obliczeń równoległych, jest wprowadzana przez użytkownika,
results - zmienna tablicowa, służy do przechowywania wyników obliczeń.

2.3.4 Funkcje

eval_fun - funkcja przetwarzająca funkcję wpisaną przez użytkownika. Określa ona matematyczną postać i oblicza jej wartość w danym punkcie. Jej argumenty to zmienna string przechowująca postać funkcji oraz zmienna float przechowująca zmienną dla której wartość funkcji ma być obliczona,
Integrate - funkcja obliczeniowa programu, oblicza za pomocą metody trapezów wartość całki oznaczonej na danym przedziale. Jej argumenty to funkcja dla której całka jest obliczana, dolna i górna granica całkowania i krok obliczeń,
frange - pomocnicza funkcja, używana do tworzeniu list zmiennych typu float.

2.3.5 Klasy

Pool - umożliwia obliczenia równoległe w prosty sposób. W skład klasy wchodzą cztery metody : Pool.apply, Pool.map, Pool.apply_async , Pool.map_async . Metody Pool.map oraz Pool.apply blokują główny program do czasu aż wszystkie procesy zostaną ukończone. Metoda Pool.apply_async która tworzy procesy w jednej chwili i zwraca wyniki jak tylko zostaną ukończone. Ta metoda jest wykorzystana w projekcie obliczeniowym.
Program dzieli przedział obliczeń wynikający z zadanych dolnej i górnej granicy całkowania na mniejsze podprzedziały, których liczba jest równa liczbie stworzonych wątków obliczeniowych. Dla każdego z podprzedziałów wywoływana jest metoda Pool.apply_async , która wykorzystuje funkcję **Integrate**. Wyniki trafiają do tablicy **results**, która po zakończeniu całkowania jest sumowana, a ta suma czyli końcowy wynik jest drukowana na standardowe wyjście/

2.3.6 Pomiar czasu działania programu

Do pomiaru czasu działania programu chciano użyć kodu służącego do tego umieszczając go w programie. Niestety kilka rozwiązań znalezionych w internecie nie działało prawidłowo, wyświetlany czas wykonywania programu był za krótki lub długi, nie miał zbyt wiele wspólnego z rzeczywistością. W tej sytuacji postanowiono czas działania programu liczyć ręcznie tzn. stoperem w smartfonie. Czasy działania programu rozważane w projekcie to od kilkunastu do kilkuset sekund, zakładając czas reakcji człowieka mierzącego czas na około jedną sekundę, błąd tak uzyskanych wyników jest akceptowalnie mały, wystarczająco mały jak na potrzeby tego projektu. Początkiem pomiaru był czas wciśnięcia klawisza Enter po wpisaniu ostatniej wartości w terminal - liczby wątków a końcem pomiaru wyświetlenie wyniku przez program.

3 Wyniki i analiza danych

3.1 Kalkulator prostych całek

Najpierw postanowiono sprawdzić czy wyniki obliczeń numerycznych całek są bliskie wynikom analitycznym. W tym celu policzono za pomocą programu całki z kilku funkcji i porównano z wynikiem dokładnym (analitycznym). Wyniki w Tabeli 1. Widzimy, że wyniki uzyskane numerycznie są bliskie wartościom analitycznym (dokładnym) i wiarygodne.

Tabela 1: Wyniki obliczeń numerycznych całek i ich wartości analityczne, podany krok całkowania i ilość wątków w obliczeniach numerycznych

L.p.	całka	wynik analityczny	wynik numeryczny	krok	wątki
1.	$\int_0^{10} x dx$	50	49.9992	0.00001	100
2.	$\int_1^5 x^3 dx$	156	155.9838	0.00001	100
3.	$\int_0^{\pi} \sin(x) dx$	2	1.99962	0.00001	100
4.	$\int_0^{\frac{\pi}{2}} \cos(x) dx$	1	0.99949	0.00001	100
5.	$\int_0^{\pi} x \sin(x) dx$	3.14159	3.14099	0.00001	100

3.2 Analiza kroku całkowania i liczby wątków na czas rozwiązania

Następnie postanowiono zbadać wpływ kroku całkowania i ilości użytych wątków na czas liczenia całki. Postanowiono liczyć taką samą całkę przy trzech różnych krokach całkowania : 0,0001 , 0,00003 , 0,00001 oraz czterech liczbach wątkóW : 10, 100, 1000, 10000.
Na potrzeby tej analizy postanowiono liczyć numerycznie całkę :

$$I = \int_0^{20\pi} x^2 \cos^2(x) dx \tag{3.2.1}$$

Wynik analityczny tej całki wynosi :

$$I = \int x^2 \cos^2(x) dx = \frac{1}{24}(4x^3 + (6x^2 - 3)\sin(2x) + 6x\cos(2x)) + C \tag{3.2.2}$$

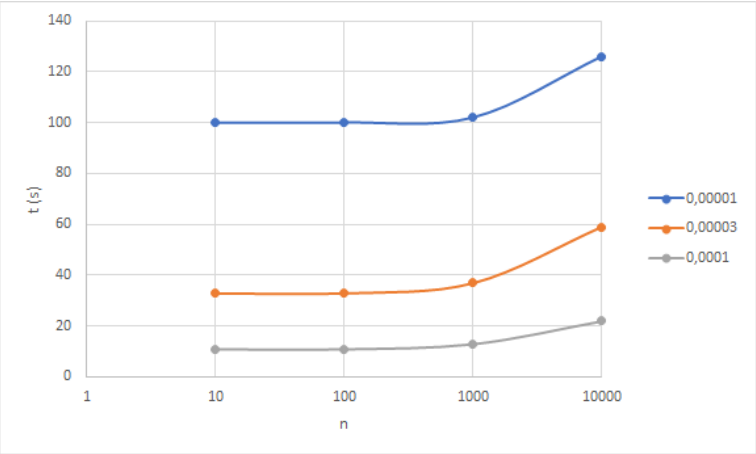
$$I = \int_0^{20\pi} x^2 \cos^2(x) dx = 41358,0 \tag{3.2.3}$$

Wyniki obliczeń numerycznych przedstawiono w Tabeli 2 i na Rysunku 4.

Tabela 2: Wyniki obliczeń numerycznych całki I dla danego kroku, liczby wątków n, podane wartości czasu działania programu t i wynik całki I

L.p.	krok	n	t(s)	I
1.	0.0001	10	11	41356.7
2.	0.0001	100	11	41356.6
3.	0.0001	1000	13	41336.9
4.	0.0001	10000	22	40810.2
5.	0.00003	10	33	41357.6
6.	0.00003	100	33	41356.2
7.	0.00003	1000	37	41350.1
8.	0.00003	10000	59	41271.1
9.	0.00001	10	100	41357.9
10.	0.00001	100	100	41357.4
11.	0.00001	1000	102	41356.7
12.	0.00001	10000	126	41336.9

Widzimy, że wyniki obliczeń numerycznych dla wszystkich wariantów kroku całkowania i liczby wątków są bliskie wartości analitycznej danej wzorem (3.2.3). Zmniejszanie kroku całkowania powoduje zwiększenie czasu działania programu. Zwiększanie liczby wątków nie skraca czasu działania programu a dla liczby wątków $n = 10000$ nawet go wydłuża. Powodem tego mogły być koszty komunikacji procesorów rosnące ze wzrostem ich liczby czy potrzeba duplikacji obliczeń na różnych procesorach. Być może czas podziału problemu na różne procesory zdominował czas potrzebny na obliczenie całki.



Rysunek 4: Wykres przedstawiający czas liczenia całki t w funkcji ilości wątków n dla trzech kroków całkowania : 0,0001 , 0,00003 , 0,00001

4 Podsumowanie

Zapoznano się z platformą Microsoft Azure. Nauczono się tworzyć tam maszyny wirtualne, uzyskiwać do nich dostęp za pomocą programu PuTTy, przesyłać pliki za pomocą WinSCP, oraz obsługiwać system Linux z poziomu terminala.
Wyniki obliczeń numerycznych całek są zadowalająco bliskie wynikom analitycznym. Zmniejszanie kroku całkowania powoduje zwiększenie czasu działania programu. Zwiększanie liczby wątków nie skraca czasu działania programu a dla liczby wątków $n = 10000$ nawet go wydłuża. Powodem tego mogły być koszty komunikacji procesorów rosnące ze wzrostem ich liczby czy potrzeba duplikacji obliczeń na różnych procesorach. Być może czas podziału problemu na różne procesory zdominował czas potrzebny na obliczenie całki.