


Paulina Grabowska 	Metody Numeryczne
Metoda prostokątów & Metoda trapezów Metoda Simpsona & Metoda Monte Carlo	

1. Wstęp

Całkowanie numeryczne to proces przybliżonego obliczania wartości całki oznaczonej funkcji na określonym przedziale. W odróżnieniu od całkowania analitycznego, które opiera się na dokładnych metodach algebraicznych, całkowanie numeryczne wykorzystuje techniki numeryczne i komputerowe do przybliżenia wyniku.

Idea całkowania numerycznego opiera się na podziale przedziału całkowania na niewielkie części, na których wartość funkcji jest przybliżana. Istnieje kilka podstawowych metod całkowania numerycznego, z których najpopularniejsze to metoda prostokątów, metoda trapezów, metoda Simpsona oraz metoda Monte Carlo.

Ważną kwestią podczas całkowania numerycznego jest także kontrola błędów numerycznych, które mogą wynikać z ograniczeń komputerowych oraz przybliżeń użytych w metodach. Dlatego istotne jest zrozumienie, jakie błędy mogą wystąpić i jak je minimalizować.

Całkowanie numeryczne znajduje zastosowanie w różnych dziedzinach nauki i techniki, szczególnie tam, gdzie nie ma dostępu do rozwiązań analitycznych lub są one zbyt skomplikowane. Dzięki wykorzystaniu metod numerycznych, możemy szybko i efektywnie obliczać wartości całek dla szerokiego zakresu funkcji, co jest kluczowe w wielu aplikacjach inżynierskich, naukowych i matematycznych.

2. Metody całkowania numerycznego

2.1. Metoda Prostokątów

Metoda prostokątów to jedna z prostszych technik numerycznego całkowania stosowana w analizie numerycznej do przybliżonego obliczania wartości całki funkcji na danym przedziale. W metodzie tej pole pod wykresem funkcji na danym przedziale dzieli się na prostokąty, których pola są sumowane, aby uzyskać przybliżoną wartość całki.

Podział przedziału odbywa się na równe podprzedziały, a wartości funkcji są obliczane w punkcie końcowym każdego podprzedziału. Następnie te wartości są mnożone przez szerokość podprzedziału i sumowane, aby uzyskać przybliżoną wartość całki.

Jeśli mamy przybliżony przedział całkowania $[a, b]$ podzielony na n równych podprzedziałów o szerokości Δx , to przybliżona wartość całki funkcji $f(x)$ na tym przedziale będzie wynosić:

$$\int_a^b f(x) dx \approx \sum_{i=0}^n f(a + i \cdot \Delta x) \cdot \Delta x \quad (1)$$

Gdzie $a + i \cdot \Delta x$ to punkt końcowy i -tego podprzedziału. Sumowanie odbywa się dla i od zera do n .

2.2. Metoda Trapezów

Metoda trapezów jest kolejną techniką całkowania numerycznego, która polega na przybliżeniu wartości całki oznaczonej funkcji na danym przedziale poprzez sumowanie obszarów trapezów, których boki przylegają do krzywej funkcji. W przeciwieństwie do metody prostokątów, gdzie obszary pod wykresem funkcji są przybliżane za pomocą prostokątów, metoda trapezów używa trapezów do bardziej precyzyjnego przybliżenia.

Idea metody trapezów opiera się na przybliżeniu krzywej funkcji na każdym podprzedziale poprzez interpolację liniową między wartościami funkcji w dwóch krańcach tego podprzedziału. Następnie pole trapezu, który powstaje poprzez połączenie tych dwóch punktów oraz pionowych linii na obu krańcach podprzedziału, jest używane jako przybliżenie pola pod krzywą funkcji na danym podprzedziale.

Formalnie, aby obliczyć przybliżoną wartość całki oznaczonej funkcji $f(x)$ na przedziale $[a, b]$ za pomocą metody trapezów, dzielimy ten przedział na n równych podprzedziałów o długości

$$\Delta x = \frac{b - a}{n} \quad (2)$$

Następnie dla każdego z tych podprzedziałów, przybliżamy pole pod krzywą funkcji za pomocą pola trapezu. Sumując te przybliżone pola trapezów na wszystkich podprzedziałach, otrzymujemy przybliżoną wartość całki:

(3)

$$\int_a^b f(x) dx \approx \frac{\Delta x}{2} [f(a) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1}) + f(b)]$$

Gdzie x_1, x_2, x_n , są punktami wewnątrz każdego z podprzedziałów. Wartości $f(a)$ i $f(b)$ są wartościami funkcji na krańcach przedziału.

Graficznie, metoda trapezów polega na przybliżeniu obszarów pod krzywą funkcji na każdym podprzedziale za pomocą trapezów, których jedna podstawa jest równoległa do osi x , a druga podstawa jest interpolacją liniową między wartościami funkcji na krańcach podprzedziału.

Im większa liczba podprzedziałów n , tym dokładniejsze będzie przybliżenie wartości całki. Metoda trapezów jest popularna ze względu na swoją prostotę i szybkość, a także dlatego, że daje dokładniejsze wyniki niż metoda prostokątów, szczególnie dla funkcji o większej zmienności.

2.3. Metoda Simpsona

Metoda Simpsona jest metodą całkowania numerycznego, która służy do przybliżania wartości całki oznaczonej funkcji na danym przedziale poprzez interpolację wielomianami drugiego stopnia (parabolami) między wartościami funkcji w trzech równoodległych punktach na każdym podprzedziale. Metoda ta jest bardziej zaawansowana niż metody prostokątów i trapezów, ponieważ używa wielomianów stopnia drugiego do interpolacji, co prowadzi do dokładniejszych wyników, szczególnie dla funkcji o gładkim przebiegu.

Idea metody Simpsona polega na tym, że na każdym podprzedziale $[x_i, x_{i+2}]$ przybliżamy krzywą funkcji wielomianem stopnia drugiego, który idealnie pasuje do trzech punktów: $(x_i, f(x_i)); (x_{i+1}, f(x_{i+1})); (x_{i+2}, f(x_{i+2}))$.

Pole pod tym wielomianem jest następnie używane jako przybliżenie pola pod krzywą funkcji na danym podprzedziale.

Formalnie, aby obliczyć przybliżoną wartość całki oznaczonej funkcji $f(x)$ na przedziale $[a, b]$ za pomocą metody Simpsona, dzielimy ten przedział na n równych podprzedziałów o długości (2). Następnie obliczamy wartość całki za pomocą wzoru:

$$\int_a^b f(x) dx \approx \frac{\Delta x}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 4f(x_{n-1}) + f(x_n)] \quad (4)$$

Gdzie $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ są równoodległymi punktami na przedziale $[a, b]$. Wartości $f(x_0)$ i $f(x_n)$ są wartościami funkcji na krańcach przedziału, a pozostałe wartości funkcji są obliczane w równoodległych punktach wewnątrz każdego podprzedziału.

Graficznie, metoda Simpsona polega na przybliżeniu obszarów pod krzywą funkcji na każdym podprzedziale za pomocą parabol, które idealnie pasują do trzech równoodległych punktów.

Im większa liczba podprzedziałów n , tym dokładniejsze będzie przybliżenie wartości całki. Metoda Simpsona jest szczególnie przydatna do przybliżania całek funkcji, które są dobrze zachowane i mają gładki przebieg.

2.4. Metoda Monte Carlo

Metoda Monte Carlo to technika numeryczna wykorzystywana do przybliżania wartości całek oraz rozwiązywania różnych problemów matematycznych i statystycznych poprzez losowanie próbek z odpowiedniego zakresu i analizowanie ich charakterystyk. Nazwa tej metody pochodzi od słynnego kasyna Monte Carlo, gdzie często wykorzystuje się element losowości w grach hazardowych.

Idea metody Monte Carlo opiera się na założeniu, że możemy estymować wartość całki oznaczonej funkcji na danym przedziale, obliczając średnią wartość funkcji dla losowo wybranych punktów wewnątrz tego przedziału, pomnożoną przez szerokość przedziału. Im więcej punktów losujemy, tym dokładniejsze będzie przybliżenie.

Kroki metody Monte Carlo są następujące: wygenerowanie losowych punktów - losujemy N punktów z zakresu wartości niezależnej zmiennej (np. osi x dla całki jednowymiarowej, lub płaszczyzny xy dla całki dwuwymiarowej) oraz obliczenie wartości funkcji. Dla każdego wylosowanego punktu, obliczamy wartość funkcji, którą chcemy całkować. Następnie obliczamy średnią wartość funkcji dla wszystkich wylosowanych punktów i mnożymy ją przez obszar (szerokość przedziału lub pole powierzchni), na którym przeprowadzamy symulację.

Matematycznie, przybliżona wartość całki za pomocą metody Monte Carlo może być wyrażona jako:

$$\int_a^b f(x) dx \approx (b - a) \cdot \frac{1}{N} \sum_{i=1}^N f(x_i) \quad (5)$$

gdzie x_i są losowymi punktami wybranymi z przedziału $[a, b]$, a N jest liczbą wylosowanych punktów.

Metoda Monte Carlo jest użyteczna w przypadkach, gdy tradycyjne metody analityczne lub numeryczne są niewykonalne lub niepraktyczne ze względu na złożoność funkcji lub wymiary problemu. Jej siła tkwi w tym, że przy odpowiednio dużej liczbie losowanych punktów, przybliżenie wartości całki staje się coraz dokładniejsze, zgodnie z prawem wielkich liczb.

3. Implementacja numeryczna

3.1. Funkcje *function* oraz *main*

Zaczęto od definicji dyrektywy preprocesora *#define*, która pozwala na definiowanie makr, które umożliwiają zastępowanie określonych fragmentów tekstu (*Fragment kodu 1*). Skorzystano z tego, ponieważ umożliwia to szybką zmianę używanego typu – domyślnie używano typu *double*, ale przyszłościowo taka definicja umożliwia późniejsze zmiany na typy zmiennych *float* lub *long double*.

```
1. #define DD double
```

Fragment kodu 1: definicja dyrektywy preprocesora *DD*

Następnie zdefiniowano funkcję o nazwie *function* (*Fragment kodu 2*), która przyjmuje argument typu *DD* o nazwie *x* i zwraca wartość typu *DD*. Funkcja ta jest zadeklarowana jako zwracająca wartość typu *DD* – którą można szybko zmienić korzystając z wcześniej zadeklarowanej dyrektywy *#define*.

Instrukcja *return*, która zwraca wartość wyrażenia $x^3 + 2$. Jest to miejsce, w którym zapisuje się dla jakiej funkcji będziemy całkować. Ta instrukcja jest centralną częścią funkcji, która określa, jakie działanie wykonuje funkcja na swoim argumencie i jaką wartość zwraca.

```
1. DD function(DD x) {  
2.     return (x*x*x + 2);  
3. }
```

Fragment kodu 2: definicja funkcji *function*

Główna funkcja *main* programu przeprowadza obliczenia całek numerycznych dla podanej funkcji na określonym przedziale i wyświetla wyniki (*Fragment kodu 3*). Zadeklarowane zostają zmienne typu *DD* *a*, *b*; oraz *int* *n*. Następuje deklaracja zmiennych *a*, *b*, które reprezentują odpowiednio początek i koniec przedziału całkowania, oraz zmiennej *n* typu *int*, która określa liczbę podprzedziałów.

Program wypisuje komunikaty celem wprowadzenia danych z poziomu konsoli. Program wykonuje obliczenia dla czterech różnych metod całkowania numerycznego (*rectangle*, *trapeze*, *simpson*, *monte_carlo*) dla podanych wartości a , b oraz n . Wyniki obliczeń są przechowywane w zmiennych *rectangle_check*, *trapeze_check*, *simpson_check*, *montecarlo_check*.

```
1. int main() {
2.     DD a, b;
3.     int n;
4.
5.     cout << "Podaj początek przedziału całkowania (a): ";
6.     cin >> a;
7.     cout << "Podaj koniec przedziału całkowania (b): ";
8.     cin >> b;
9.     cout << "Podaj liczbę podprzedziałów (n): ";
10.    cin >> n;
11.
12.    DD rectangle_check = rectangle(a, b, n);
13.    DD trapeze_check = trapeze(a, b, n);
14.    DD simpson_check = simpson(a, b, n);
15.    DD montecarlo_check = monte_carlo(a, b, n);
16.
17.    cout << endl << endl << endl;
18.    cout << "rectangle result: " << rectangle_check << endl << endl;
19.    cout << "trapeze result: " << trapeze_check << endl << endl;
20.    cout << "simpson result: " << simpson_check << endl << endl;
21.    cout << "monte carlo result: " << montecarlo_check << endl << endl;
22.
23.    return 0;
24. }
```

Fragment kodu 3: funkcja *main*

3.2. Funkcja *rectangle*

Funkcja *rectangle* przyjmuje trzy argumenty: a (dolny limit całkowania), b (górny limit całkowania) oraz n (liczbę podprzedziałów) (Fragment kodu 4). Na początku obliczana jest szerokość każdego podprzedziału dx jako różnicę między b i a , podzieloną przez liczbę podprzedziałów n . Inicjalizujemy także zmienną *result* jako 0, która będzie przechowywać sumę obszarów prostokątów.

Następnie pętla *for* iteruje przez wszystkie podprzedziały, od 0 do $n-1$. W każdej iteracji obliczana wartość x_i - punktu w środku danego podprzedziału. Dzieje się to poprzez dodanie do a początku podprzedziału i -tego, przesunięcia o połowę szerokości podprzedziału dx . Czyli x_i jest równy $a + i * dx + dx$.

W każdej iteracji dodawane są wartość funkcji w punkcie x_i do sumy *result*. Funkcja *function(x_i)* jest funkcją, która jest całkowana. Na koniec, po zakończeniu pętli sumę *result*

mnożona jest przez szerokość podprzedziału dx , co daje przybliżoną wartość całki, i zwracany jest ten wynik jako wartość bezwzględna, celem uniknięcia ujemnego wyniku.

```
1. DD rectangle(DD a, DD b, int n) {
2.     DD dx = (b - a) / n;
3.     DD result = 0.0;
4.
5.     for (int i = 0; i < n; ++i) {
6.         DD xi = a + i * dx + dx;
7.         result += function(xi);
8.     }
9.
10.    return abs(dx * result) ;
11. }
12.
```

Fragment kodu 4: definicja funkcji *rectangle*

3.3. Funkcja *trapeze*

Ten fragment kodu implementuje funkcję *trapeze*, która oblicza przybliżoną wartość całki oznaczonej funkcji na danym przedziale za pomocą metody trapezów (*Fragment kodu 5*). Następuje deklaracja funkcji *trapeze*, która przyjmuje trzy argumenty: a (początek przedziału całkowania), b (koniec przedziału całkowania) oraz n (liczbę podprzedziałów)

Następnie obliczana jest szerokość każdego podprzedziału na podstawie wartości a , b i n . Szerokość ta jest używana do przybliżenia obszaru pod krzywą funkcji na każdym z podprzedziałów.

Inicjalizowana jest zmienna *result*, która będzie przechowywać sumę obszarów trapezów. Wartość początkowa *result* jest obliczana jako średnia wartość funkcji na krańcach przedziału, podzielona przez 2.0, co odpowiada średniej wartości podstawy trapezu.

Pętla *for* iteruje przez wszystkie podprzedziały, począwszy od pierwszego, ponieważ wartość dla $i = 0$ została już dodana w inicjalizacji zmiennej *result*. W każdej iteracji pętli dodawana jest wartość funkcji dla punktu wewnątrz każdego podprzedziału, przesuniętego od początku przedziału o odpowiednią wartość $i \cdot dx$.

Obliczona suma obszarów trapezów jest mnożona przez szerokość podprzedziału dx , aby uzyskać przybliżoną wartość całki, która jest zwracana przez funkcję. Ta funkcja implementuje metodę trapezów, gdzie obszar pod krzywą funkcji na każdym podprzedziale jest przybliżany jako trapez, a całkowita wartość całki jest sumą tych przybliżeń.

```

1. DD trapeze(DD a, DD b, int n) {
2.     DD dx = (b - a) / n;
3.     DD result = (function(a) + function(b)) / 2.0;
4.
5.     for (int i = 1; i < n; ++i) {
6.         result += function(a + i * dx);
7.     }
8.
9.     return abs(dx * result);
10. }

```

Fragment kodu 5: definicja funkcji *trapeze*

3.4. Funkcja *simpson*

Ten kod implementuje funkcję *simpson*, która oblicza przybliżoną wartość całki oznaczonej funkcji na danym przedziale za pomocą metody Simpsona (*Fragment kodu 6*). funkcja typu, typu *DD*, przyjmuje trzy argumenty: *a* (początek przedziału całkowania), *b* (koniec przedziału całkowania) oraz *n* (liczbę podprzedziałów).

Zaczyna się od obliczenia szerokości każdego podprzedziału na podstawie wartości *a*, *b* i *n*. Szerokość ta jest używana do przybliżenia obszaru pod krzywą funkcji na każdym z podprzedziałów. Zainicjalizowana zmienna *result*, która będzie przechowywać sumę wartości funkcji na krańcach przedziału. Wartość ta odpowiada sumie wartości funkcji na krańcach przedziału, która jest mnożona przez 2 w kolejnych krokach obliczeń.

Pierwsza pętla *for* iteruje przez wszystkie nieparzyste indeksy *i* od 1 do *n-1*. Dla każdej iteracji, obliczana jest wartość *x* wewnątrz podprzedziału, a następnie wartość funkcji w tym punkcie jest dodawana do *result* pomnożona przez 4, ponieważ te punkty odpowiadają głównym punktom w metodzie Simpsona.

Druga pętla *for* iteruje przez wszystkie parzyste indeksy *i* od 2 do *n-2*. Dla każdej iteracji, obliczana jest wartość *x* wewnątrz podprzedziału, a następnie wartość funkcji w tym punkcie jest dodawana do *result* pomnożona przez 2, ponieważ te punkty odpowiadają pozostałym punktom w metodzie Simpsona.

Obliczona suma wartości funkcji na podprzedziałach jest mnożona przez $dx/3$, co odpowiada metodzie Simpsona, gdzie obszar pod krzywą jest przybliżany za pomocą parabol, a całkowita wartość całki jest obliczana jako $dx/3$ mnożona przez sumę wartości funkcji na krańcach przedziału i punktach wewnątrz przedziału.


```

1. DD simpson(DD a, DD b, int n) {
2.     DD dx = (b - a) / n;
3.     DD result = function(a) + function(b);
4.
5.     for (int i = 1; i < n; i += 2) {
6.         DD x = a + i * dx;
7.         result += 4 * function(x);
8.     }
9.
10.    for (int i = 2; i < n - 1; i += 2) {
11.        DD x = a + i * dx;
12.        result += 2 * function(x);
13.    }
14.
15.    return abs(result * dx / 3.0);
16. }

```

Fragment kodu 6: definicja funkcji *simpson*

3.5. Funkcja *monte_carlo*

Ten fragment kodu implementuje funkcję *monte_carlo*, która oblicza przybliżoną wartość całki oznaczonej funkcji na danym przedziale za pomocą metody Monte Carlo (*Fragment kodu 7*). Następuje inicjalizacja generatora liczb pseudolosowych za pomocą funkcji *srand*, co pozwala na generowanie losowych liczb za każdym razem, gdy program jest uruchamiany. Funkcja *time(NULL)* służy jako ziarno generatora liczb pseudolosowych, które zmienia się w czasie, co zapewnia większą losowość generowanych liczb.

Inicjalizowana jest zmienna *result*, która będzie przechowywać sumę wartości funkcji dla punktów losowo wybranych z przedziału $[a, b]$. Pętla *for*, która iteruje n razy, generując n losowych punktów na przedziale $[a, b]$ i obliczając wartość funkcji w tych punktach.

W każdej iteracji pętli generowany jest losowy punkt x_r na przedziale $[a, b]$. Zmienna x_r jest obliczana jako suma a oraz losowej wartości z przedziału $[0, 1]$ przeskalowanej do przedziału $[0, b-a]$. Wszystkie wylosowane punkty wypisywane są na ekranie, celem ich kontroli. Ten krok nie jest zazwyczaj wymagany w funkcji Monte Carlo, ale może być użyteczny do debugowania, aby zobaczyć wygenerowane punkty.

Do zmiennej *result* dodawana jest wartość funkcji $function(x_r)$ w wygenerowanym punkcie x_r . Obliczona suma wartości funkcji jest mnożona przez szerokość przedziału $b-a$, a następnie dzielona przez liczbę wygenerowanych punktów n . Wynik ten stanowi przybliżoną wartość całki oznaczonej funkcji na przedziale $[a, b]$, zgodnie z metodą Monte Carlo.

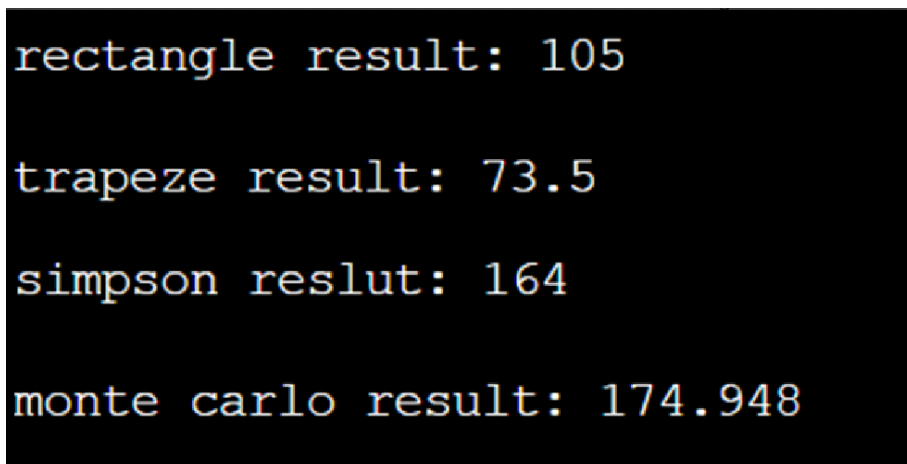
```

1. DD monte_carlo(DD a, DD b, int n) {
2.     srand(time(NULL));
3.
4.     DD result = 0.0;
5.
6.     for (int i = 0; i < n; ++i) {
7.         DD xr = a + (b - a) * rand() / RAND_MAX;
8.         cout << xr << " ";
9.         result += function(xr);
10.    }
11.
12.    return abs((b - a) * result / n);
13. }
14.

```

Fragment kodu 7: definicja funkcji *monte_carlo*

Po zaimplementowaniu wszystkich funkcji sprawdzono się poprawność na podstawie przykładów podanych na zajęciach i otrzymano wyniki poprawne (*Rys. 1*), co pozwala na stwierdzenie, że wszystkie funkcje zostały zaimplementowane poprawnie.



```

rectangle result: 105
trapeze result: 73.5
simpson reslut: 164
monte carlo result: 174.948

```

Rys. 1: zrzut ekranu konsoli

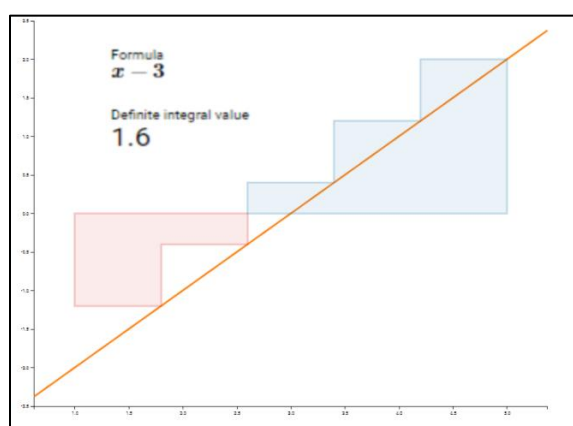
4. Testy jednostkowe

4.1. Testy metody prostokątów

4.1.1. Funkcja liniowa

Do tego testu wykorzystano funkcję o wzorze: $f(x) = x - 3$.

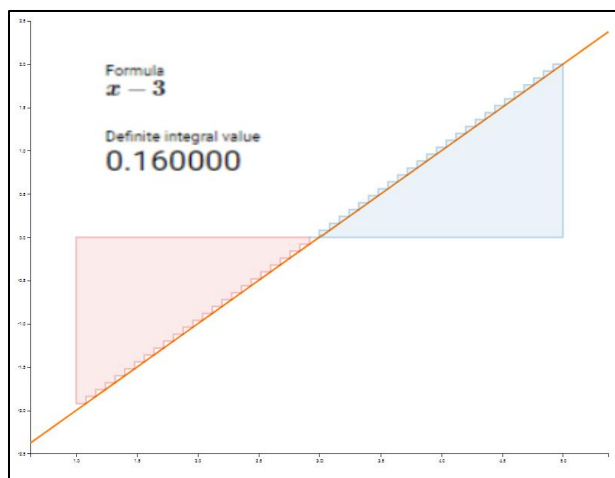
Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys 2a, 2c, 2e) oraz załączono wyniki z konsoli (Rys. 2b, 2d, 2f).



Rys. 2a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji liniowej obliczone w programie PlanetCalc

```
rectangle result: 1.6
```

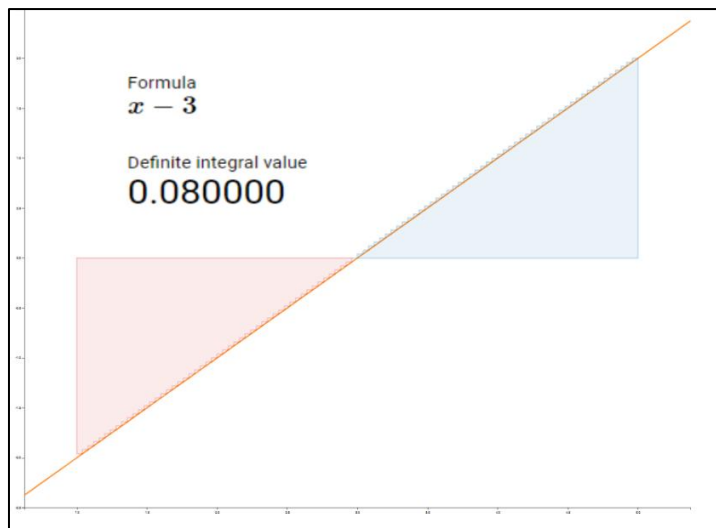
Rys. 2b Wynik dla liczby n równej 5 w teście dla funkcji liniowej



Rys. 2c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji liniowej obliczone w programie PlanetCalc

```
rectangle result: 0.16
```

Rys. 2d Wynik dla liczby n równej 50 w teście dla funkcji liniowej



Rys. 2e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji liniowej obliczone w programie PlanetCalc

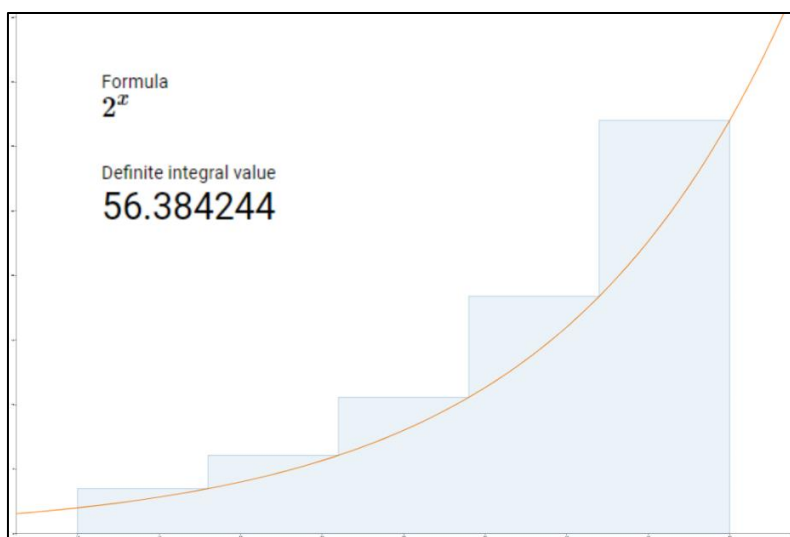
```
rectangle result: 0.08
```

Rys. 2f Wynik dla liczby n równej 100 w teście dla funkcji liniowej

4.1.2. Funkcja wykładnicza

Do tego testu wykorzystano funkcję o wzorze: $f(x) = 2^x$.

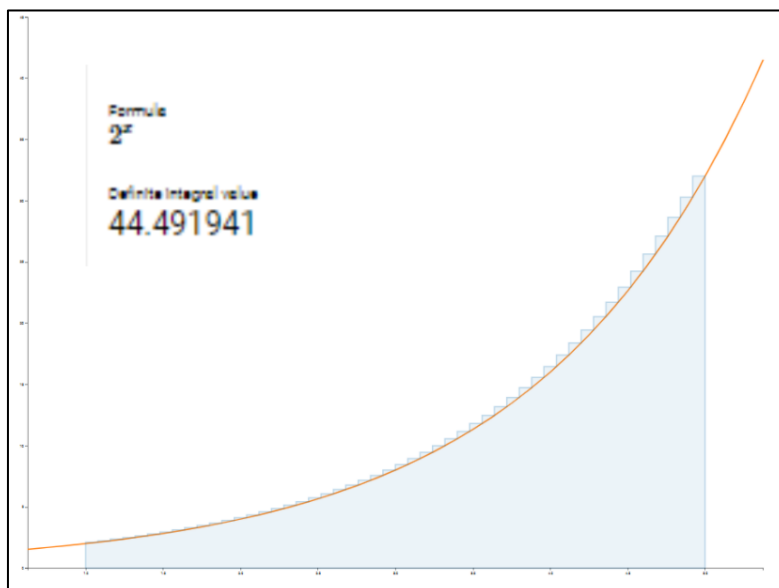
Ustawiono przedział na wartości $[1, 5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys. 3a, 3c, 3e) oraz załączono wyniki z konsoli (Rys. 3b, 3d, 3f).



Rys. 3a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

rectangle result: 56.3842

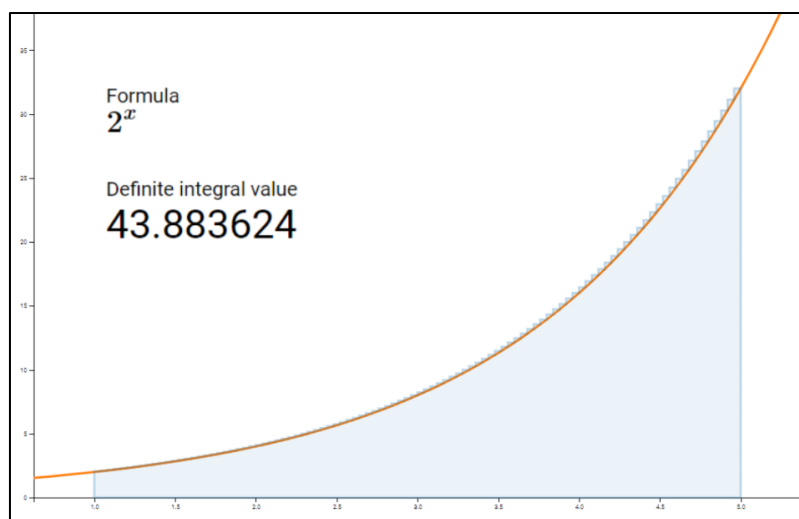
Rys. 3b Wynik dla liczby n równej 5 w teście dla funkcji wykładniczej



Rys. 3c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

rectangle result: 44.4919

Rys. 3d Wynik dla liczby n równej 50 w teście dla funkcji wykładniczej



Rys. 3e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

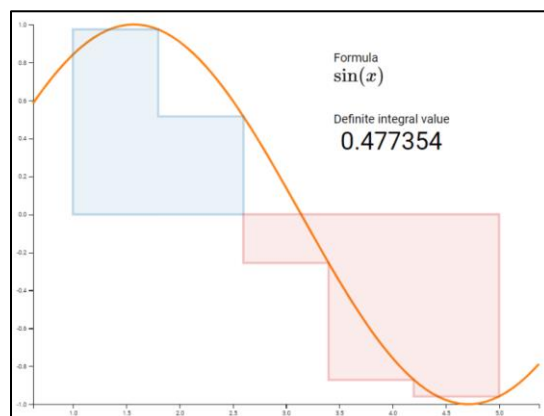
```
rectangle result: 43.8836
```

Rys. 3f Wynik dla liczby n równej 100 w teście dla funkcji wykładniczej

4.1.3. Funkcja sinusoidalna

Do tego testu wykorzystano funkcję o wzorze: $f(x) = \sin x$.

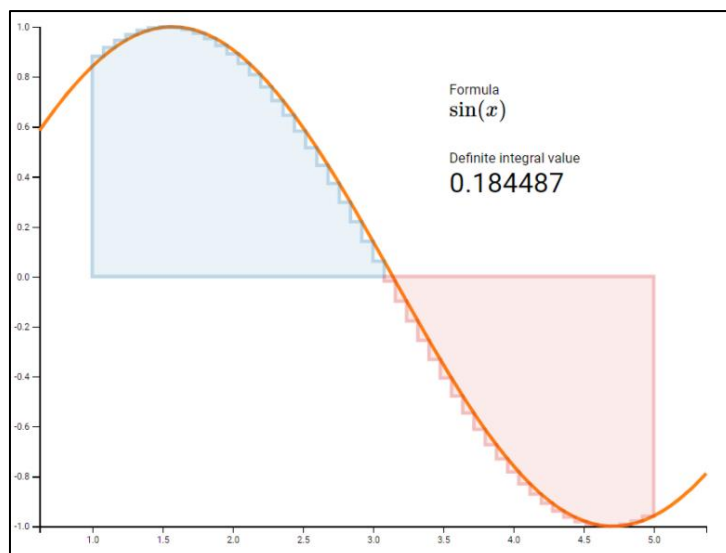
Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys. 4a, 4c, 4e) oraz załączono wyniki z konsoli (Rys. 4b, 4d, 4f).



Rys. 4a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji sinusoidalnej obliczone w programie PlanetCalc

```
rectangle result: 0.477354
```

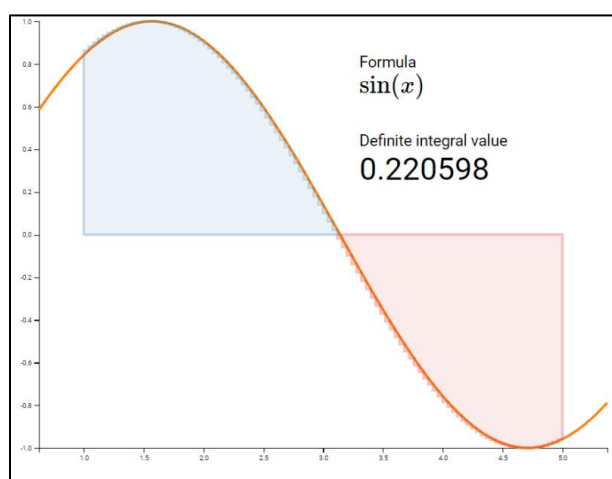
Rys. 4b Wynik dla liczby n równej 5 w teście dla funkcji sinusoidalnej



Rys. 4c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji sinusoidalnej
obliczone w programie PlanetCalc

```
rectangle result: 0.184487
```

Rys. 4d Wynik dla liczby n równej 50 w teście dla funkcji sinusoidalnej



Rys. 4e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji sinusoidalnej
obliczone w programie PlanetCalc

```
rectangle result: 0.220598
```

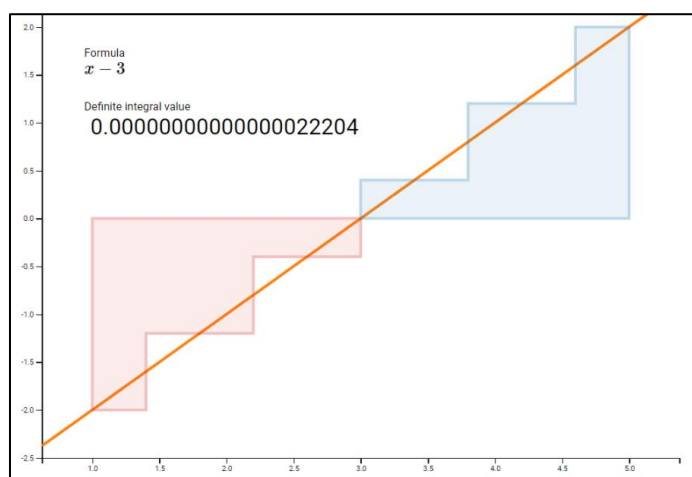
Rys. 4f Wynik dla liczby n równej 100 w teście dla funkcji sinusoidalnej

4.2. Testy metody trapezów

4.2.1. Funkcja liniowa

Do tego testu wykorzystano funkcję o wzorze: $f(x) = x - 3$

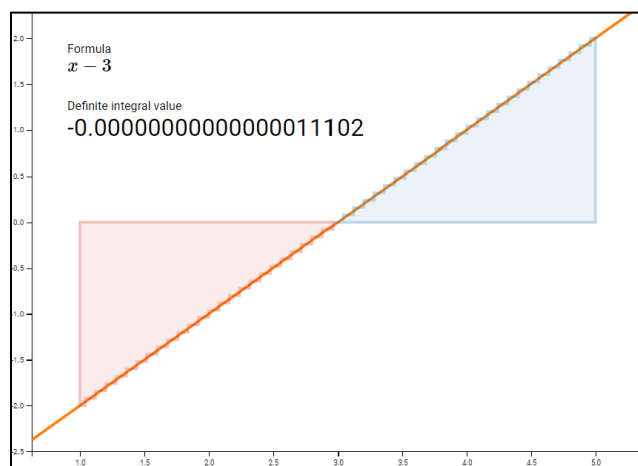
Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys 5a, 5c, 5e) oraz załączono wyniki z konsoli (Rys. 5b, 5d, 5f).



Rys. 5a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji liniowej obliczone w programie PlanetCalc

```
trapeze result: 5.32907e-16
```

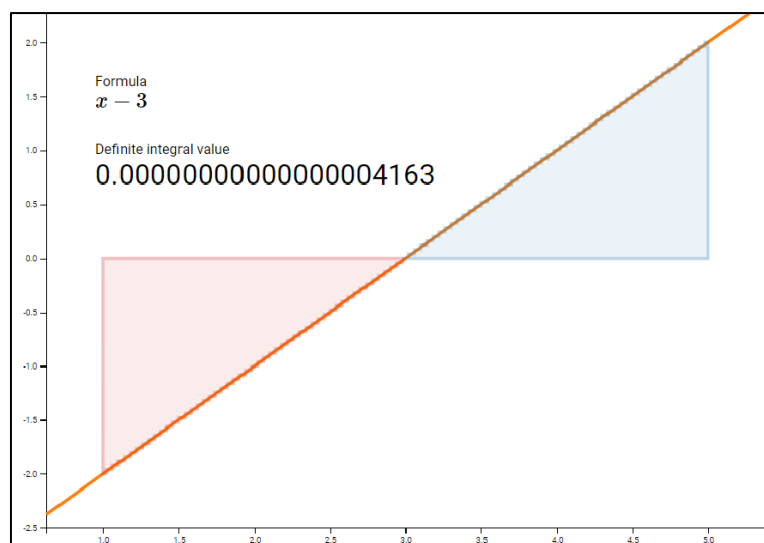
Rys. 5b Wynik dla liczby n równej 5 w teście dla funkcji liniowej



Rys. 5c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji liniowej obliczone w programie PlanetCalc


```
trapeze result: 0
```

Rys. 5d Wynik dla liczby n równej 50 w teście dla funkcji liniowej



Rys. 5e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji liniowej
obliczone w programie PlanetCalc

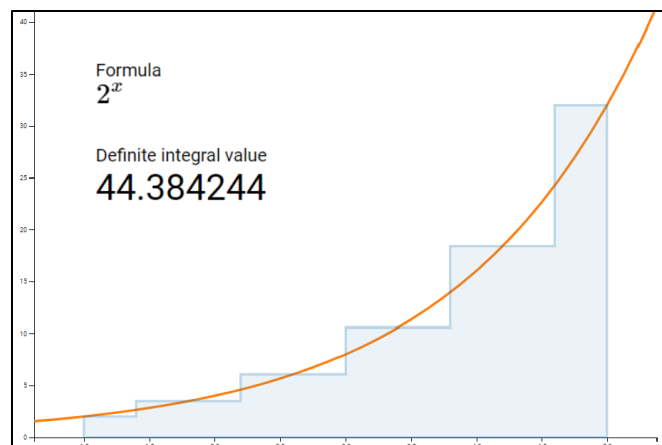
```
trapeze result: 4.9738e-16
```

Rys. 5f Wynik dla liczby n równej 100 w teście dla funkcji liniowej

4.2.2. Funkcja wykładnicza

Do tego testu wykorzystano funkcję o wzorze: $f(x) = 2^x$

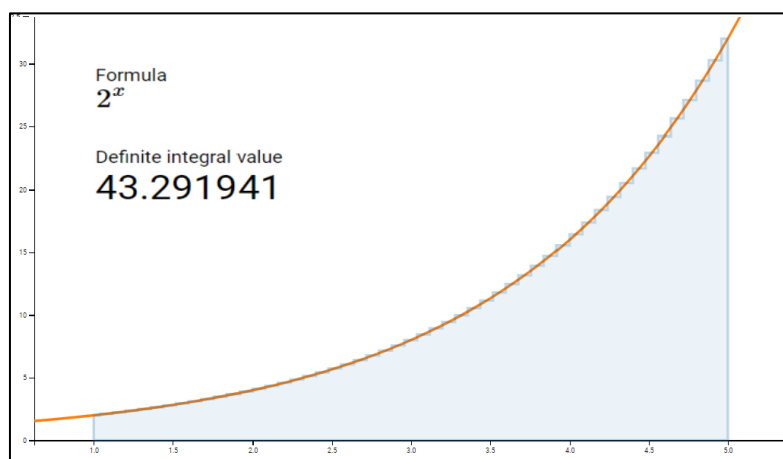
Ustawiono przedział na wartości $[1, 5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (*Rys 6a, 6c, 6e*) oraz załączono wyniki z konsoli (*Rys. 6b, 6d, 6f*).



Rys. 6a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

trapeze result: 44.3842

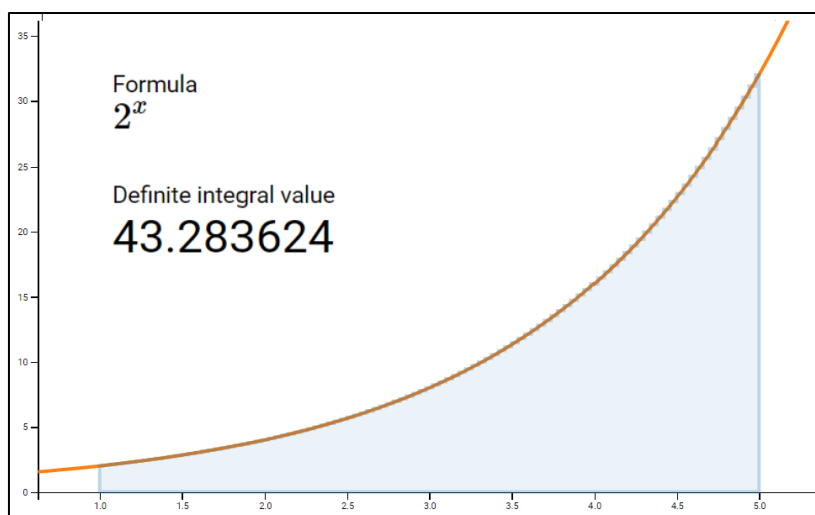
Rys. 6b Wynik dla liczby n równej 5 w teście dla funkcji wykładniczej



Rys. 6c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

trapeze result: 43.2919

Rys. 6d Wynik dla liczby n równej 50 w teście dla funkcji wykładniczej



Rys. 6e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

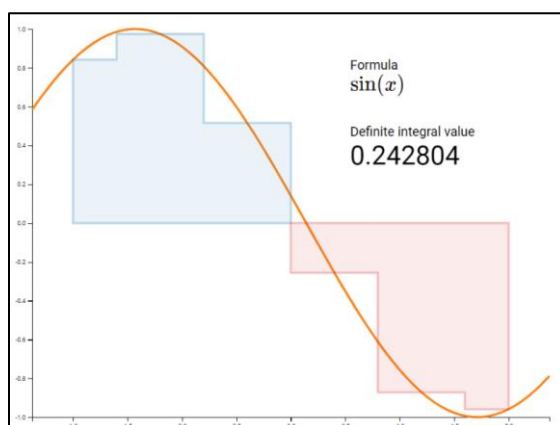
trapeze result: 43.2836

Rys. 6f Wynik dla liczby n równej 100 w teście dla funkcji wykładniczej

4.2.3. Funkcja sinusoidalna

Do tego testu wykorzystano funkcję o wzorze: $f(x) = \sin x$

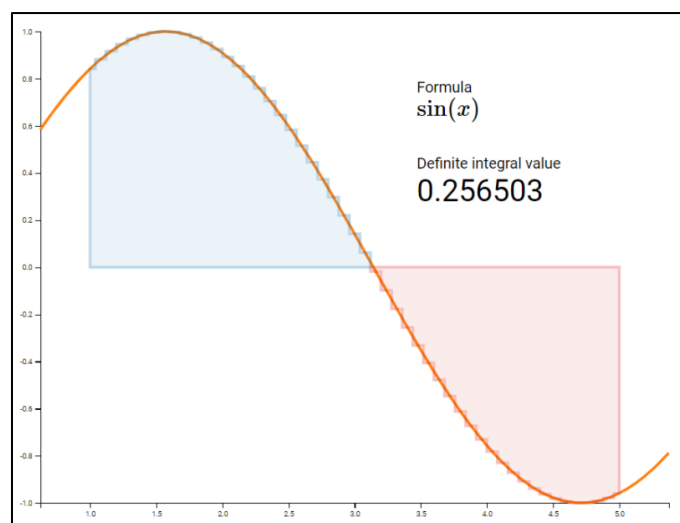
Ustawiono przedział na wartości $[1, 5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys 7a, 7c, 7e) oraz załączono wyniki z konsoli (Rys. 7b, 7d, 7f).



Rys. 7a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji sinusoidalnej obliczone w programie PlanetCalc

```
trapeze result: 0.242804
```

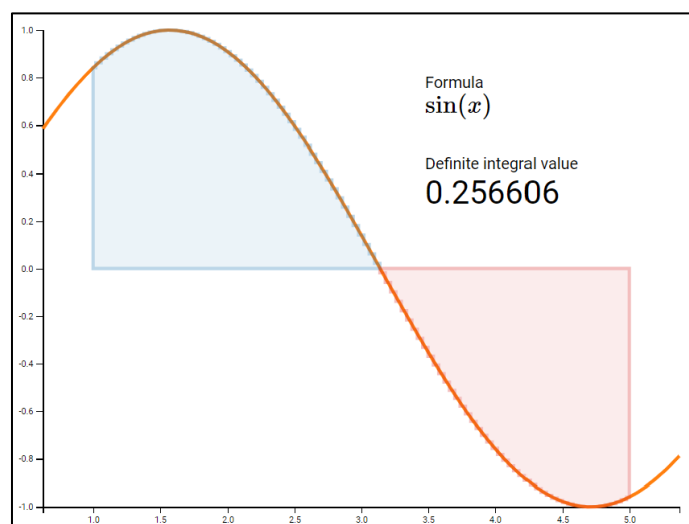
Rys. 7b Wynik dla liczby n równej 5 w teście dla funkcji sinusoidalnej



Rys. 7c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji sinusoidalnej
obliczone w programie PlanetCalc

```
trapeze result: 0.256503
```

Rys. 7d Wynik dla liczby n równej 50 w teście dla funkcji sinusoidalnej



Rys. 7e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji sinusoidalnej
obliczone w programie PlanetCalc

```
trapeze result: 0.256606
```

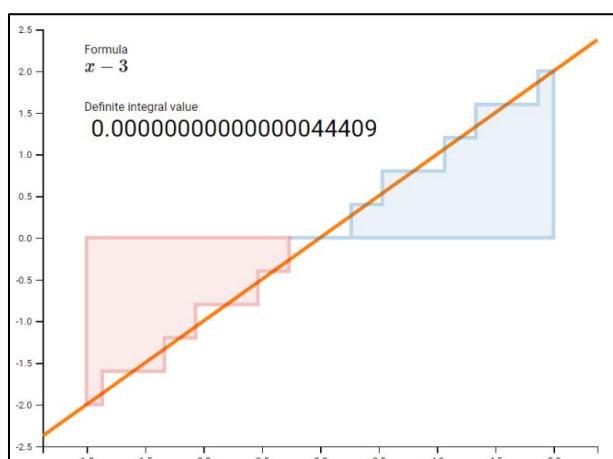
Rys. 7f Wynik dla liczby n równej 100 w teście dla funkcji sinusoidalnej

4.3. Testy metody Simpsona

4.3.1. Funkcja liniowa

Do tego testu wykorzystano funkcję o wzorze: $f(x) = x - 3$

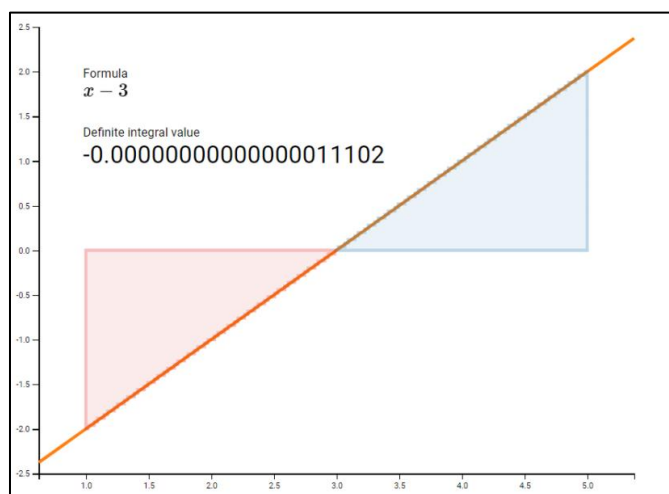
Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys. 8a, 8c, 8e) oraz załączono wyniki z konsoli (Rys. 8b, 8d, 8f).



Rys. 8a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji liniowej obliczone w programie PlanetCalc

```
simpson reslut: 1.06667
```

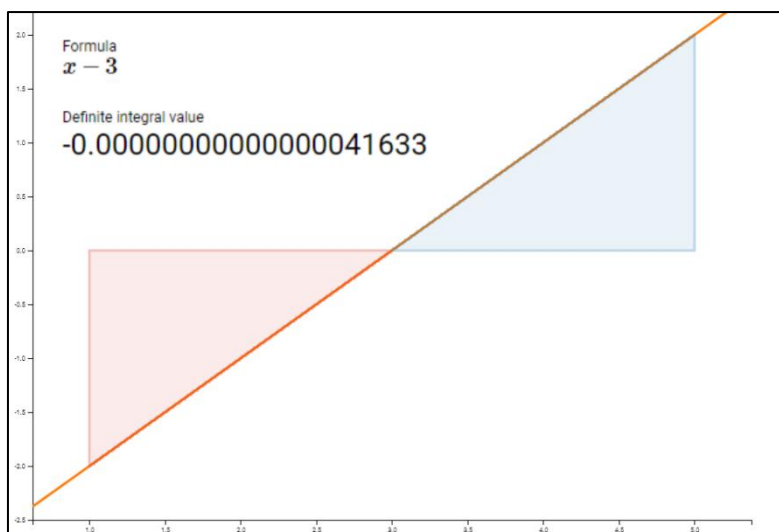
Rys. 8b Wynik dla liczby n równej 5 w teście dla funkcji liniowej



Rys. 8c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji liniowej obliczone w programie PlanetCalc

```
simpson reslut: 2.84217e-16
```

Rys. 8d Wynik dla liczby n równej 50 w teście dla funkcji liniowej



Rys. 8e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji liniowej
obliczone w programie PlanetCalc

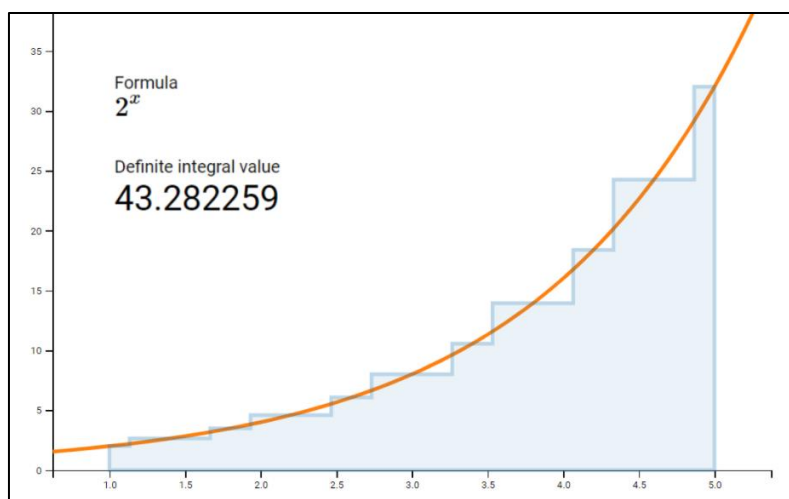
```
simpson reslut: 0
```

Rys. 8f Wynik dla liczby n równej 100 w teście dla funkcji liniowej

4.3.2. Funkcja wykładnicza

Do tego testu wykorzystano funkcję o wzorze: $f(x) = 2^x$

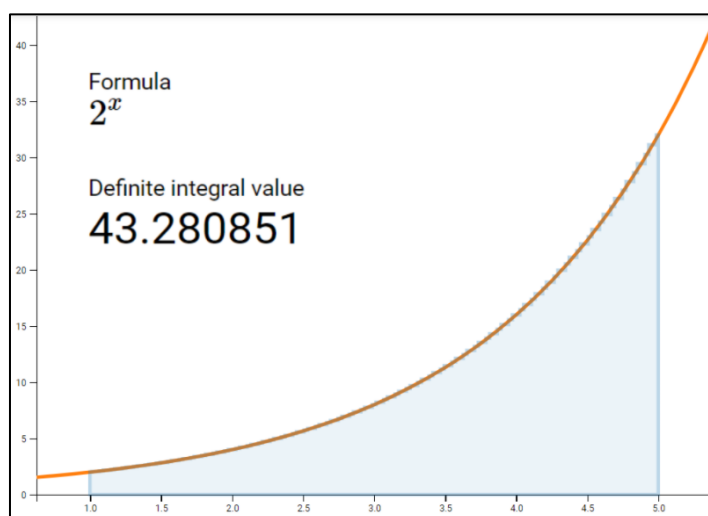
Ustawiono przedział na wartości $[1, 5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys. 9a, 9c, 9e) oraz załączono wyniki z konsoli (Rys. 9b, 9d, 9f).



Rys. 9a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

```
simpson reslut: 27.2743
```

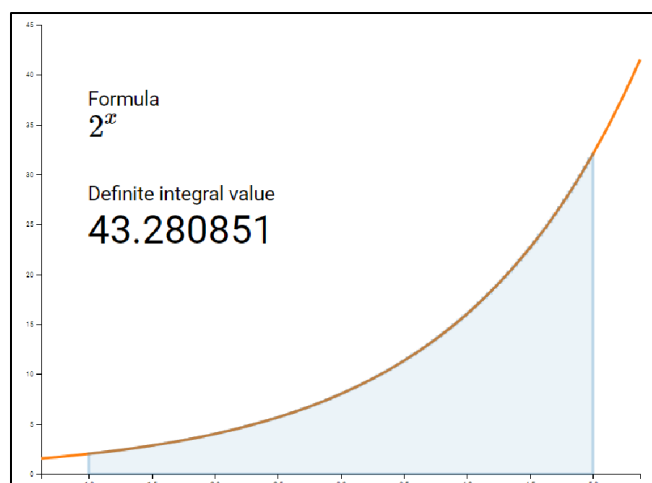
Rys. 9b Wynik dla liczby n równej 5 w teście dla funkcji wykładniczej



Rys. 9c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

```
simpson reslut: 43.2809
```

Rys. 9d Wynik dla liczby n równej 50 w teście dla funkcji wykładniczej



Rys. 9e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji wykładniczej obliczone w programie PlanetCalc

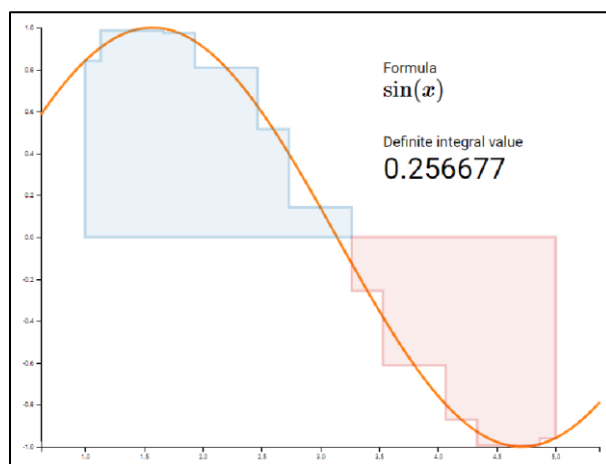
```
simpson reslut: 43.2809
```

Rys. 9f Wynik dla liczby n równej 100 w teście dla funkcji wykładniczej

4.3.3. Funkcja sinusoidalna

Do tego testu wykorzystano funkcję o wzorze: $f(x) = \sin x$

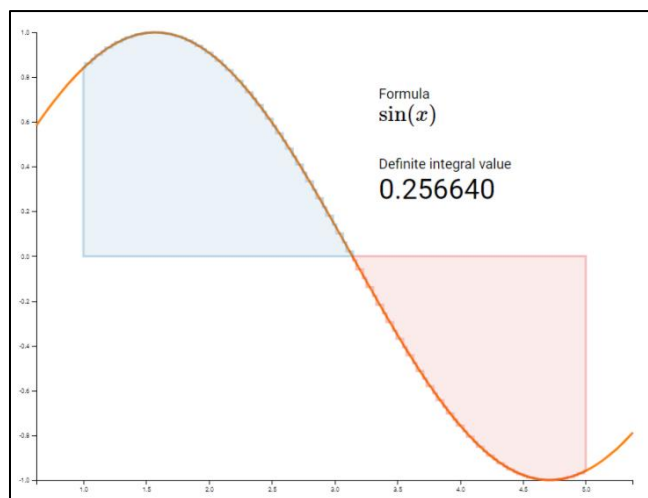
Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymane wyniki dla każdej liczby n zwizualizowano za pomocą programu PlanetCalc, gdzie również sprawdzono poprawność obliczonego przez program wyniku (Rys. 10a, 10c, 10e) oraz załączono wyniki z konsoli (Rys. 10b, 10d, 10f).



Rys.10a Przewidywane wyniki dla liczby n równej 5 w teście dla funkcji sinusoidalnej obliczone w programie PlanetCalc


```
simpson reslut: 1.00981
```

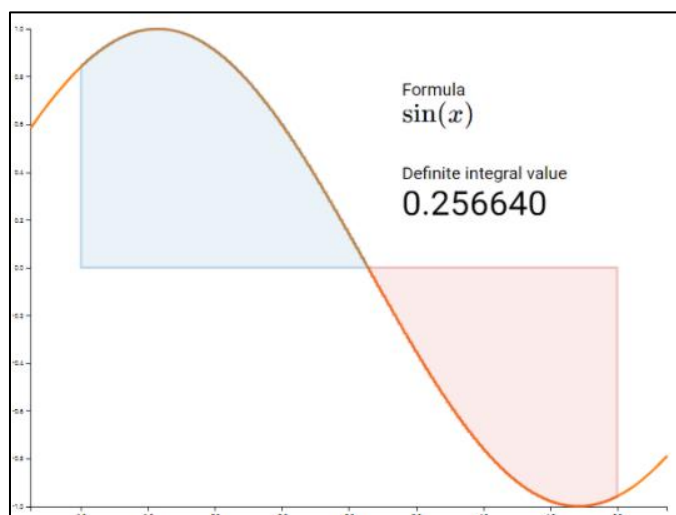
Rys. 10b Wynik dla liczby n równej 5 w teście dla funkcji sinusoidalnej



Rys. 10c Przewidywane wyniki dla liczby n równej 50 w teście dla funkcji sinusoidalnej obliczone w programie PlanetCalc

```
simpson reslut: 0.25664
```

Rys. 10d Wynik dla liczby n równej 50 w teście dla funkcji sinusoidalnej



Rys. 10e Przewidywane wyniki dla liczby n równej 100 w teście dla funkcji sinusoidalnej obliczone w programie PlanetCalc

```
simpson reslut: 0.25664
```

Rys. 10f Wynik dla liczby n równej 100 w teście dla funkcji sinusoidalnej

4.4. Testy metody Monte Carlo

4.4.1. Funkcja liniowa

Do tego testu wykorzystano funkcję o wzorze: $f(x) = x - 3$

Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymany wynik sprawdzono w programie WolframAlphaa (Rys. 11a) oraz załączono wyniki z konsoli (Rys. 11b, 11c, 11d).

$$\int_1^5 (x - 3) dx = 0$$

Rys. 11a Przewidywane wyniki w teście dla funkcji liniowej obliczone w programie Wolfram Alpha

```
monte carlo result: 2.05902
```

Rys. 11b Wynik dla liczby n równej 5 w teście dla funkcji liniowej

```
monte carlo result: 1.12119
```

Rys. 11c Wynik dla liczby n równej 50 w teście dla funkcji liniowej

```
monte carlo result: 0.43725
```

Rys. 11d Wynik dla liczby n równej 100 w teście dla funkcji liniowej

4.4.2. Funkcja wykładnicza

Do tego testu wykorzystano funkcję o wzorze: $f(x) = 2^x$

Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymany wynik sprawdzono w programie WolframAlphaa (Rys. 12a) oraz załączono wyniki z konsoli (Rys. 12b, 12c, 12d).

$$\int_1^5 2^x dx = \frac{30}{\log(2)} \approx 43.281$$

Rys. 12a Przewidywane wyniki w teście dla funkcji wykładniczej obliczone w programie Wolfram Alpha

```
monte carlo result: 38.9607
```

Rys. 12b Wynik dla liczby n równej 5 w teście dla funkcji wykładniczej

```
monte carlo result: 43.6329
```

Rys. 12c Wynik dla liczby n równej 50 w teście dla funkcji wykładniczej

```
monte carlo result: 43.6121
```

Rys. 12d Wynik dla liczby n równej 100 w teście dla funkcji wykładniczej

4.4.3. Funkcja sinusoidalna

Do tego testu wykorzystano funkcję o wzorze: $f(x) = \sin x$

Ustawiono przedział na wartości $[1,5]$ i sprawdzono otrzymane wyniki dla różnej liczby n : odpowiednio 5, 50 i 100. Otrzymany wynik sprawdzono w programie WolframAlphaa (*Rys. 13a*) oraz załączono wyniki z konsoli (*Rys. 13b, 13c, 13d*).

$$\int_1^5 \sin(x) dx = \cos(1) - \cos(5) \approx 0.25664$$

Rys. 12a Przewidywane wyniki w teście dla funkcji sinusoidalnej obliczone w programie Wolfram Alpha

```
monte carlo result: 0.464755
```

Rys. 13b Wynik dla liczby n równej 5 w teście dla funkcji sinusoidalnej

```
monte carlo result: 0.352958
```

Rys. 13b Wynik dla liczby n równej 50 w teście dla funkcji sinusoidalnej

```
monte carlo result: 0.401911
```

Rys. 13b Wynik dla liczby n równej 100 w teście dla funkcji sinusoidalnej

5. Opracowanie wyników

Wyniki sugerują, że zaimplementowana metoda prostokątów działa poprawnie i uzyskuje przewidywane wyniki (*Tabela 1*). Natomiast jak widać po rozbieżności między tymi wynikami, nie jest to metoda zbyt dokładna, dająca miarodajne wyniki. Wraz ze wzrostem liczby podprzedziałów otrzymana wartość jest coraz bliższa dokładnemu wynikowi całki oznaczonej. Wszystkie wyniki otrzymane pokrywają się z tymi, które sugeruje PlantCalc dla tej metody. W przypadku funkcji liniowej wraz z e wzrostem liczby podprzedziałów wynik zbliża się do poprawnej wartości całki oznaczonej. Dla funkcji wykładniczej zachodzi podobna zależność. W przypadku funkcji sinusoidalnej można zauważyć podobną prawidłowość – im większa liczba podprzedziałów tym bliżej wyniku całki oznaczonej.

Tabela 1: Zestawienie wyników metody prostokątów

Funkcja	Wartość całki oznaczonej	Liczba podprzedziałów n	Wartość oczekiwana	Wartość otrzymana
f. liniowa	0	5	1,6	1,6
		50	0,16	0,16
		100	0,08	0,08
f. wykładnicza	43,281	5	56,3842	56,3842
		50	44,4919	44,4919
		100	43,8836	43,8836
f. sinusoidalna	0,25664	5	0,477354	0,477354
		50	0,184487	0,184487
		100	0,220598	0,220598

Wyniki sugerują, że metoda trapezów może być skuteczna w przybliżaniu wartości całek funkcji (Tabela 2). W przypadku funkcji liniowej pojawiają się rozbieżności w błędach – wyniki otrzymane nie pokrywają się z tymi otrzymanymi w programie PlanetCalc, chociaż różnice te nie są duże, a przez dużą ujemną potęgę są o wiele bardziej zbliżone do zera, które jest poprawnym wynikiem, niż wyniki otrzymane metodą prostokątów. Można więc wysunąć wniosek, że dla funkcji liniowej metoda ta może być bardziej podatna na błędy w porównaniu do funkcji wykładniczej i sinusoidalnej. Dla tych ostatnich dwóch funkcji, metoda trapezów zdaje się dawać dokładniejsze wyniki.

Tabela 2: Zestawienie wyników metody trapezów

Funkcja	Wartość całki oznaczonej	Liczba podprzedziałów n	Wartość oczekiwana	Wartość otrzymana
f. liniowa	0	5	$2,2204 \cdot 10^{-16}$	$5,32907 \cdot 10^{-16}$
		50	$1,1102 \cdot 10^{-16}$	0
		100	$4,163 \cdot 10^{-17}$	$4,9728 \cdot 10^{-16}$
f. wykładnicza	43,281	5	44,3942	44,3842
		50	43,2919	43,2919
		100	43,2836	43,2836
f. sinusoidalna	0,25664	5	0,242804	0,242804
		50	0,256503	0,256503
		100	0,256606	0,256606

Wyniki sugerują, że metoda Simpsona jest skuteczniejsza w przybliżaniu całek niż metoda prostokątów czy trapezów (Tabela 3). Należy jednak zwrócić uwagę na to, że w przypadku funkcji liniowej metoda ta zdaje się nie do końca sprawdzać – chociaż wyniki odbiegają od tych, przewidywanych przez program PlanetCalc, to są zbliżone do wyniku wyjściowego – liczba podprzedziałów ma tu jednak istotne znaczenie. W przypadku zarówno funkcji wykładniczej jak i sinusoidalnej, można zauważyć, że jedynie pierwszy wynik zupełnie odbiega od wszelkich przewidywań, natomiast pozostałe dwa pokrywają się zarówno z przepowiedniami programu PlanetCalc, jak i z dokładną wartością całki oznaczonej. W obliczy zestawionych wyników można poddać w wątpliwość poprawność zaimplementowanego algorytmu.

Tabela 3: Zestawienie wyników metody Simpsona

Funkcja	Wartość całki oznaczonej	Liczba podprzedziałów n	Wartość oczekiwana	Wartość otrzymana
f. liniowa	0	5	$4,4409 \cdot 10^{-16}$	1,06667
		50	$1,1102 \cdot 10^{-16}$	$2,84217 \cdot 10^{-16}$
		100	$4,1633 \cdot 10^{-16}$	0
f. wykładnicza	43,281	5	43,282259	27,2743
		50	43,2809	43,2809
		100	43,2809	43,2809
f. sinusoidalna	0,25664	5	0,256677	1,00981
		50	0,256640	0,25664
		100	0,256640	0,25664

Wyniki sugerują, że metoda Monte Carlo nie jest bardzo dokładną metodą całkowania numerycznego (*Tabela 4*). W przypadku funkcji liniowej i sinusoidalnej wyniki są dość rozbieżne i jedynie można je uznać za zbliżone do wyniku całki oznaczonej w przypadku funkcji wykładniczej. Można jednak w przypadku tej metody wysnuć wniosek, że metoda Monte Carlo zwiększa swoją dokładność wraz ze wzrostem liczby wylosowanych punktów – chociaż wyniki zestawione w tabeli tego nie oddają, to sam fakt istnienia tej losowości prowadzi nas do wniosku, że ta metoda zadziała jedynie w momencie użycia bardzo dużej ilości punktów (rzędu tysiąca)

Tabela 4: Zestawienie wyników metody Monte Carlo

Funkcja	Wartość całki oznaczonej	Liczba podprzedziałów n	Wartość otrzymana
f. liniowa	0	5	2,05902
		50	1,12119
		100	0,43725
f. wykładnicza	43,281	5	38,9307
		50	43,6329
		100	43,6121
f. sinusoidalna	0,25664	5	0,464755
		50	0,352958
		100	0,401911

6. Wnioski

W przypadku metod prostokątów i trapezów, zwiększanie liczby podprzedziałów prowadzi zazwyczaj do zwiększenia dokładności przybliżenia całki. Im więcej podprzedziałów, tym bliżej przybliżenie będzie do wartości dokładnej. Metoda Simpsona, wykorzystująca parabole do przybliżania obszaru pod krzywą funkcji, może być jeszcze bardziej dokładna przy mniejszej liczbie podprzedziałów w porównaniu do metod prostokątów i trapezów. W metodzie Monte Carlo, dokładność wyniku nie zależy od liczby podprzedziałów, ale od liczby wygenerowanych punktów. Im więcej punktów, tym dokładniejsze przybliżenie wartości całki, zgodnie z prawem wielkich liczb.

Metoda prostokątów jest najprostszą w implementacji, ale wymaga większej liczby podprzedziałów, aby uzyskać dokładne wyniki. Metoda trapezów jest bardziej precyzyjna niż metoda prostokątów i zazwyczaj daje lepsze wyniki przy tej samej liczbie podprzedziałów, ale nadal może być czasochłonna dla bardzo małych wartości n . Metoda Simpsona może być bardziej efektywna czasowo niż metoda trapezów dla uzyskania dokładniejszych wyników przy tej samej liczbie podprzedziałów. Metoda Monte Carlo może być bardzo efektywna czasowo, ponieważ nie wymaga podziału przedziału na podprzedziały ani interpolacji funkcji. Jednak liczba losowych punktów musi być wystarczająco duża, aby uzyskać dokładne wyniki.

Metody prostokątów i trapezów są stosowane w przypadkach, gdy istnieje potrzeba szybkiego przybliżenia wartości całki z funkcji. Metoda Simpsona jest często stosowana, gdy wymagana jest większa dokładność niż w przypadku metod prostokątów i trapezów, ale nadal chcemy zachować umiarkowaną prostotę obliczeń. Metoda Monte Carlo jest przydatna w przypadkach, gdy inne metody są niewykonalne lub niepraktyczne, na przykład w przypadku złożonych funkcji, wielowymiarowych całek lub problemów statystycznych. Jest również często stosowana w symulacjach i analizie danych.

Każda z tych metod ma swoje własne zalety i ograniczenia, które należy wziąć pod uwagę przy wyborze odpowiedniej metody do konkretnego zadania. W praktyce, dobór metody zależy od potrzeb, złożoności problemu oraz dostępności zasobów obliczeniowych.

7. Źródła

Wykłady z Metod Numerycznych autorstwa dr hab. Danuty Szeligi

Prezentacja Metody Numeryczne. Całkowanie numeryczne – metoda prostokątów, trapezów, Simpsona i Monte Carlo autorstwa dr. Hab. Inż. Marcina Hojnego.

Wikipedia – artykuły o całkowaniu numerycznym, metodzie prostokątów, metodzie trapezów, metodzie Simpsona i metodzie Monte Carlo

Program PlantCalc: <https://planetcalc.com/5494/#calculator6472>

Program WolframAlpha: <https://www.wolframalpha.com/>