Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Inżynierii Metali i Informatyki Przemysłowej

Sprawozdanie z Laboratorium: Całkowanie Numeryczne

Przedmiot: Metody Numeryczne

Kierunek: Inżynieria Obliczeniowa

Autor: Filip Rak

Prowadzący przedmiot: dr hab. inż. Marcin Hojny

Data: 5 kwietnia 2024

Numer lekcji: 5

Grupa laboratoryjna: 4

Wstęp teoretyczny

Całkowaniem numerycznym nazywamy metodę numeryczną, której celem jest obliczanie przybliżonych wartości całek oznaczonych. Całkę oznaczoną interpretujemy jako pole powierzchni między wykresem funkcji $f(x)$ w pewnym przedziale $[a, b]$, a osią odciętych.

$$y = \int_a^b f(x)dx \tag{1}$$

Przybliżone wartości całek numerycznych możemy uzyskać poprzez podzielenie całki na n przedziałów, obliczenie wartości naszej funkcji w tych przedziałach i zsumowanie uzyskanych wyników.

Opis metod całkowania numerycznego

Metoda prostokątów

Do jednych z najprostszych metod należy metoda prostokątów. Metodę możemy opisać następującym wzorem:

$$dx[f(x_i) + f(x_{i+1}) + \cdots + f(x_n)] \tag{2}$$

Przy czym x_i do x_n są kolejnymi pozycjami prostokątów na osi X. dx jest szerokością naszego prostokąta i możemy ją obliczyć w poniższy sposób:

$$dx = \frac{b - a}{n} \tag{3}$$

Dla b i a będącymi kolejno, końcem i początkiem przedziału całkowania a n ilością wytworzonych prostokątów.

Poszczególne argumenty x będziemy wyliczać stosując następujący wzór

$$x_i = a + i\ dx \tag{4}$$

Metoda trapezów

Metoda trapezów jest mocno zbliżona do metody prostokątów. Różnicą jest zastosowanie trapezów w miejscu prostokątów, które w znacznie większym stopniu dopasowują się do wykresu funkcji, oferują poprawę w dokładności aproksymacji wyniku. Metodę opisuje następujący wzór:

$$dx\left[\frac{1}{2}f(x_1) + f(x_2) + f(x_3) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)\right] \quad (5)$$

Wartość dx obliczamy wykorzystując wzór (3). x_1 do x_n są kolejnymi pozycjami prostokątów na osi X.

Metoda Simpsona

Metoda Simpsona jest kolejnym krokiem naprzód pod względem przybliżenia wartości całki oznaczonej. W odróżnieniu od poprzednich metod, wykorzystuje ona parabole zamiast linii prostych, co znacznie poprawia dokładność aproksymacji.

W metodzie Simpsona obszar pod krzywą jest dzielony na parzystą liczbę równych podprzedziałów, a następnie na każdych dwóch takich podprzedziałach definiuje się parabolę przechodzącą przez wartości funkcji w końcach tych podprzedziałów i w ich środkowym punkcie. Limitacją metody Simpsona jest to, że wymaga ona parzystej ilości przedziałów do poprawnego działania. Metodę opisujemy poniższym wzorem:

$$\frac{dx}{3}\left[f(x_0) + 4\sum_{i=1,3,5\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,6\ldots}^{n-2} f(x_i) + f(x_n)\right] \quad (6)$$

Przy czym dx obliczamy ze wzoru (3), pierwsza suma obejmuje nieparzyste indeksy a druga suma obejmuje indeksy parzyste. x_0 do x_n to punkty podziału na przedziale całkowania.

Metoda Monte Carlo

Metoda Monte Carlo jest techniką opierająca się na statystycznych próbkach losowych, która jest używana do przybliżania skomplikowanych całek, dla których trudno zastosować tradycyjne metody. Metoda polega na wylosowaniu n punktów z całkowanego przedziału, które posłużą jako argumenty x dla całkowanej funkcji. Metodę możemy opisać następującym wzorem:

$$(b-a)\,\frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{7}$$

Przy czym, b i a są kolejno końcem i początkiem przedziału całkowania a n liczbą losowych próbek. Suma obejmuje wyliczone wartości funkcji f dla losowo wybranych argumentów.

Implementacja

Implementacja powyższych metod została przeprowadzona w języku C++ w postaci czterech niezależnych funkcji.

Cechy wspólne funkcji

Wszystkie cztery funkcje reprezentujące zaimplementowane techniki przyjmują poniższy zestaw argumentów:

- `double (*f)(double)` - wskaźnik do całkowanej funkcji.
- `double a` - początek przedziału całkowania.
- `double b` - koniec przedziału całkowania.
- `int n` - liczba podziałów obszaru całkowania lub wylosowanych punktów w przypadku metody Monte Carlo.

Dodatkową cechą wspólną funkcji jest zwracany typ danych `double`. Funkcja całkowana jest reprezentowana w kodzie w następujący sposób:

```
double function(double x)
{
    return pow(x, 3) + 2;
}
```

Definicja przedstawionej funkcji będzie modyfikowana w zależności od potrzeb na etapie przeprowadzania testów.

Implementacja metody prostokątów

Metoda prostokątów została zaimplementowana w postaci funkcji `rectangleRule`. Jej pełna definicja jest następująca:

```
double rectangleRule(double (*f)(double), double a, double b, int n)
{
    double dx = (b - a) / n;

    double result = 0;
    for (int i = 1; i <= n; i++)
    {
        double x = a + i * dx;
        result += f(x);
    }

    return result * dx;
}
```

Omówienie funkcji `rectangleRule`

Działając zgodnie ze wzorem (2), funkcja na początku swojej pracy wylicza szerokości pojedynczego prostokąta, jest to realizowane z użyciem wzoru (3).

```
double dx = (b - a) / n;
```

Zmienna **`result`** służy do przechowania sumy wszystkich wyliczonych y.

```
double result = 0;
```

Pętla **`for`** jest wykorzystywana do obliczenia sumy wszystkich y. Iteracje wykonywane są `n` razy dla przedziału $[1, n]$. Dla każdej iteracji, najpierw wyliczany jest argument **`x`**, z wykorzystaniem wzoru (4). Następnie, obliczony argument, przekazujemy badanej funkcji. Wartości funkcji są sumowane i przechowywane w zmiennej **`result`**.

Suma wartości funkcji `f` zostaje pomnożona przez szerokość **`dx`**, zgodnie ze wzorem (2),

```
for (int i = 1; i <= n; i++)
{
    double x = a + i * dx;
    result += f(x);
}
```

Po czym wynik jest zwracany.

```
return result * dx;
```

Implementacja metody trapezów

Metoda trapezów została zaimplementowana w postaci funkcji **`trapezoidalRule`**. Jej pełna definicja jest następująca:

```
double trapezoidalRule(double (*f)(double), double a, double b, int n)
{
    double dx = (b - a) / n;

    double result = (f(a) + f(b)) / 2.0;
    for (int i = 1; i < n; i++)
    {
        double x = a + i * dx;
        result += f(x);
    }

    return result * dx;
}
```

Omówienie funkcji `trapezoidalRule`

Podobnie jak w przypadku metody prostokątów, przed przejściem do dalszych obliczeń wymagana jest znajomość szerokości pojedynczego podpodziału obszaru pod funkcją. Ponownie obliczamy tą szerokość wzorem (3).

```
double dx = (b - a) / n;
```

Tworzymy zmienną na przechowywanie sumy wyników. Zgodnie ze wzorem (5), wartość y_i dla x_0 i x_n musi zostać pomnożona przez $\frac{1}{2}$. x_o i x_n to efektywnie początek i koniec przedziału całkowania a i b.

```
double result = (f(a) + f(b)) / 2.0;
```

Zadaniem pętli jest zsumowanie wyników funkcji. Iteracje wykonywane są na przedziale $[1, n)$, wykluczając początek i koniec przedziału całkowania a i b. Argumenty x obliczane są przy pomocy wzoru (4).

```
for (int i = 1; i < n; i++)
{
    double x = a + i * dx;
    result += f(x);
}
```

Zgodnie ze wzorem (5), suma jest następnie mnożona przez dx i zwracana

```
return result * dx;
```

Implementacja metody Simpsona

Metoda została zaimplementowana w postaci funkcji **`simpsonRule`**. Jej pełna definicja jest następująca:

```
double simpsonRule(double (*f)(double), double a, double b, int n)
{
    if (n % 2 != 0)
        throw std::invalid_argument("EXCEPTION: n has to be divisible by 2");

    double dx = (b - a) / n;

    double result = f(a) + f(b);
    for (int i = 1; i < n; i++)
    {
        double x = a + i * dx;
        if (i % 2 == 0)
            result += 2 * f(x);
        else
            result += 4 * f(x);
    }

    return result * dx / 3;
}
```

Omówienie funkcji `simpsonRule`

Istotnym pierwszym krokiem jest upewnienie się, że ilość obszarów jest parzysta, w innym wypadku funkcja wywołuje wyjątek i przerywa swoją pracę.

```
if (n % 2 != 0)
    throw std::invalid_argument("EXCEPTION: n has to be divisible by 2");
```

W przypadku parzystej ilości segmentów funkcja przechodzi do obliczenia szerokości pojedynczego obszaru wykorzystując wzór (3).

```
double dx = (b - a) / n;
```

Zmienna **`result`** będzie przechowywać wyniki sumowania. Inicjalizujemy ją sumą wartości brzegowych, zgodnie ze wzorem (6).

```
double result = f(a) + f(b);
```

W pętli dokonujemy zsumowania wartości funkcji dla argumentów wewnętrznych, zgodnie ze wzorem (6). Pierwszym krokiem jest wyliczenie argumentu `x`, co robimy wykorzystując wzór (4). Następnie dokonujemy obliczenia wartości funkcji dla otrzymanego argumentu, zależnie od tego czy nasz segment jest parzysty, mnożymy wynik przez 2, w innym wypadku mnożymy go przez 4.

```
for (int i = 1; i < n; i++)
{
    double x = a + i * dx;
    if (i % 2 == 0)
        result += 2 * f(x);
    else
        result += 4 * f(x);
}
```

Zwracamy nasz wynik po przemnożeniu sumy przez $\frac{dx}{3}$.

```
return result * dx / 3;
```

Implementacja metody Monte Carlo

Metoda Monte Carlo została zaimplementowana w postaci funkcji `monteCarlo`. Jej pełna definicja jest następująca:

```
double monteCarlo(double (*f)(double), double a, double b, int n)
{
    srand(time(NULL));

    double avg = 0;
    for (int i = 0; i < n; i++)
    {
        double x = a + (double)rand() / RAND_MAX * (b - a);
        avg += f(x);
    }

    avg /= n;

    return avg * abs(b - a);
}
```

Omówienie funkcji `monteCarlo`

Funkcja w pierwszej kolejności inicjalizuje generator liczb pseudolosowych, który będzie wykorzystywany w losowaniu.

```
srand(time(NULL));
```

Zmienna `avg`, będzie przechowywać średnią wartość funkcji wyliczoną na podstawie wylosowanych punktów:

```
double avg = 0;
```

W pętli dokonujemy sumowania wartości funkcji. W pierwszej kolejności z wykorzystaniem funkcji **rand**, losujemy wartość znajdującą się w przedziale $[a, b]$. Następnie dla uzyskanego argumentu wyliczamy wartość funkcji i dodajemy ją do zmiennej avg.

```
for (int i = 0; i < n; i++)
{
    double x = a + (double)rand() / RAND_MAX * (b - a);
    avg += f(x);
}
```

Dzielimy uzyskaną sumę w celu wyliczenia średniej.

```
avg /= n;
```

Średnią mnożymy przez szerokość przedziału całkowania i zwracamy.

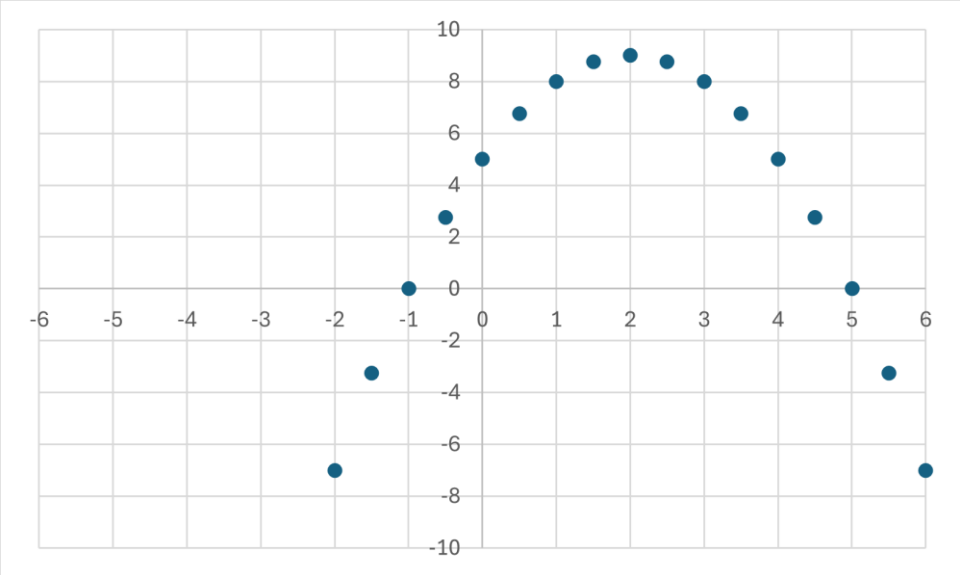
```
return avg * abs(b - a);
```

Testy na wybranych przykładach

Skuteczność zaimplementowanych metod została przetestowana na tle starannie wyselekcjonowanych przykładów. Metody zostały przetestowane dla różnych funkcji z różną ilością obszarów lub losowanych punktów w metodzie Monte Carlo n. W celach wizualizacyjnych, części wyników pokrywające się z wartością referencyjną zostały zaznaczone kolorem zielonym.

Funkcja kwadratowa

$$f(x) = -x^2 + 4x + 5$$



Wykres 1

Przedział:$[1, 4]$

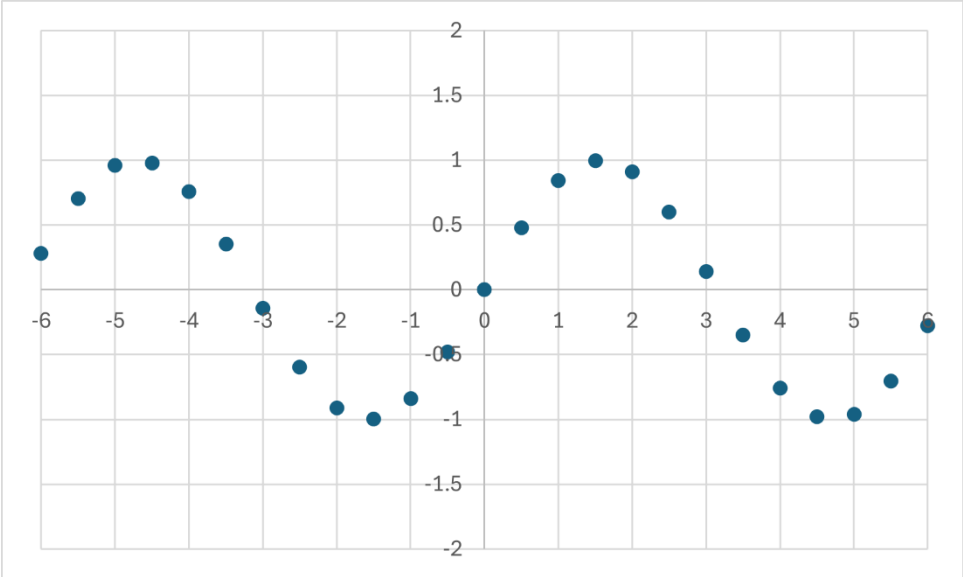
Wynik integral-calculator.com: 24

Wyniki				
$n = 1000$	$n = 150$	$n = 50$	$n = 10$	**Metoda**
23.99	23.96	23.90	23.50	*Prostokątów*
24	23.99	23.99	23.95	*Trapezów*
24	24	24	24	*Simpsona*
23.96	24.18	23.57	23.59	*Monte Carlo*

Tabela 1

Funkcja trygonometryczna

$$f(x) = \sin(x)$$



Wykres 2

Przedział: $[0, \pi]$

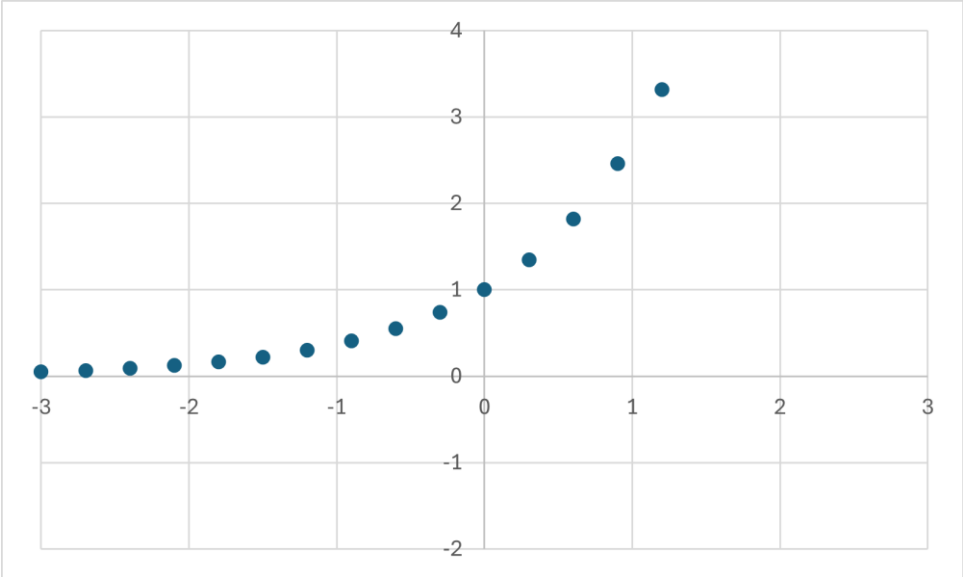
Wynik integral-calculator.com: 1.5403

Wyniki				
$n = 1000$	$n = 150$	$n = 50$	$n = 10$	**Metoda**
1.5394	1.5342	1.5220	1.4443	*Prostokątów*
1.5403	1.5402	1.5400	1.5344	*Trapezów*
1.5403	1.5403	1.5403	1.5403	*Simpsona*
1.4760	1.5769	1.4091	1.3600	*Monte Carlo*

Tabela 2

Funkcja wykładnicza

$$f(x) = e^x$$



Wykres 3

Przedział:$[0, 2]$

Wynik integral-calculator.com: 6.3890

Wyniki				
$n = 1000$	$n = 150$	$n = 50$	$n = 10$	**Metoda**
6.3954	6.4317	6.5176	7.0492	*Prostokątów*
6.3890	6.3891	6.3899	6.4103	*Trapezów*
6.3890	6.3890	6.3890	6.3891	*Simpsona*
6.2826	6.3968	6.5709	6.3813	*Monte Carlo*

Tabela 3

Opracowanie wyników

Testy zaimplementowanych metod całkowania numerycznego przeprowadzone zostały dla trzech różnych typów funkcji: kwadratowej, trygonometrycznej oraz wykładniczej, przy użyciu różnej liczby podziałów przedziału n. Wyniki uzyskane przez te metody zostały porównane z wartościami referencyjnymi otrzymanymi za pomocą integral-calculator.com, co posłużyło jako punkt odniesienia dla oceny dokładności każdej z metod.

Funkcja kwadratowa: $f(x) = -x^2 + 4x + 5$

- **Metoda prostokątów**: Wyniki zbliżały się do wartości referencyjnej (24) z rosnącą liczbą podziałów n, osiągając 23.99 przy $n = 1000$. Pokazuje to dość wysoką dokładność tej metody dla funkcji kwadratowej na badanym przedziale.
- **Metoda trapezów**: Ta metoda wykazała wyjątkowo wysoką dokładność nawet przy małej liczbie podziałów, finalnie osiągając wartość dokładnie równą referencyjnej (24).
- **Metoda Simpsona**: Metoda ta wykazała perfekcyjną zgodność z wynikiem referencyjnym na wszystkich poziomach podziałów, co potwierdza jej skuteczność dla funkcji kwadratowej.
- **Metoda Monte Carlo**: Wyniki wykazują pewne wahania, które są typowe dla metody statystycznej. Najlepsza aproksymacja (23.96) została osiągnięta przy $n = 1000$, jednak metoda wykazuje większą zmienność wyników w zależności od liczby próbek.

Funkcja trygonometryczna: $f(x) = \sin(x)$

- **Metoda prostokątów**: Zauważalny spadek dokładności z mniejszą liczbą podziałów n, osiągając 1.5394 przy $n = 1000$ w porównaniu do referencyjnego 1.5403.
- **Metoda trapezów**: Wyniki były bardzo bliskie wartości referencyjnej nawet przy niskim n, osiągając 1.5403 przy $n = 1000$.
- **Metoda Simpsona**: Dokładność tej metody była perfekcyjna we wszystkich testach, co jest oczekiwane przy całkowaniu funkcji, która ma regularny okres.
- **Metoda Monte Carlo**: Podobnie jak w poprzednim teście, wyniki wykazują zmienność, wynikająca z losowej natury metody. Największa rozbieżność od wartości referencyjnej wystąpiła przy $n = 10$, wynosząc 1.3600.

Funkcja wykładnicza: $f(x) = e^x$

- **Metoda prostokątów**: Metoda ta przejawiała systematyczną tendencję do przeszacowania wartości całki, szczególnie przy niskich wartościach n. Doskonale pokazuje to słabość metody, zwłaszcza w przypadkach stromych funkcji, wynikająca ze stosowania mało dokładnego obszaru w postaci prostokąta.
- **Metoda trapezów**: Ta metoda zapewniała wyniki w wyjątkowo ciasnym zakresie wokół wartości referencyjnej (6.3890), co wskazuje na wysoką dokładność.
- **Metoda Simpsona**: Tak jak w przypadku metody trapezów, metoda Simpsona zapewniała wyjątkową precyzję, z idealną zgodnością dla n powyżej 10.
- **Metoda Monte Carlo**: Znów zaobserwowano zmienność wyników; najlepsze przybliżenie (6.3968) uzyskano dla $n = 150$, aczkolwiek wyniki były generalnie mniej dokładne w porównaniu z innymi metodami.

Podsumowanie

Metody trapezów i Simpsona wykazały wyjątkową precyzję na przestrzeni wszystkich testów i funkcji. Metoda prostokątów była generalnie dokładna, ale w niektórych przypadkach przeszacowywała wartości. Metoda Monte Carlo wykazywała większą zmienność, co jest charakterystyczne dla jej probabilistycznej natury, jednak przy większych wartościach n była w stanie zbliżyć się do wartości referencyjnych.

Wnioski dotyczące skuteczności i dokładności każdej z metod całkowania numerycznego potwierdzają ich użyteczność w odpowiednich warunkach, z metodami Simpsona i trapezów na czele pod względem dokładności dla testowanych funkcji. Metoda Monte Carlo, mimo swojej zmienności, nadal może być użyteczna w sytuacjach, gdzie inne metody są trudne do zastosowania, na przykład w wielowymiarowych całkach lub kiedy funkcja nie jest łatwa do całkowania analitycznie.