Metody obliczeniowe w nauce i technice

Adam Naumiec
Kwiecień 2023

Laboratorium

Całkowanie numeryczne II

Spis treści

1. Treść zadania

Obliczyć przybliżoną wartość całki:

$$\int_{-\infty}^{\infty} e^{-x^2} cos(x) dx$$

(a) Przy pomocy złożonych kwadratur (prostokątów, trapezów, Simpsona),
(b) Przy pomocy całkowania adaptacyjnego,
(c) Przy pomocy kwadratury Gaussa-Hermite'a, obliczając wartości węzłów i wag.

Porównać wydajność dla zadanej dokładności.

2. Rozwiązanie

2.1. Wartość dokładna

Obliczamy dokładna wartość zadanej całki za pomocą narzędzi matematycznych:

$$\int_{-\infty}^{\infty} e^{-x^2} cos(x) dx = \frac{\sqrt{\pi}}{\sqrt[4]{e}} \approx 1{,}380388.$$

Możemy wykorzystać fakt, że funkcja całkowana jest funkcją parzystą (dowód oczywisty) i zapisać:

$$\int_{-\infty}^{\infty} e^{-x^2} cos(x) dx = 2 \int_{0}^{\infty} e^{-x^2} cos(x) dx.$$

2.2. (a) Złożone kwadratury

2.2.1. Kwadratura złożona prostokątów

$$2 \int_{0}^{\infty} e^{-x^2} cos(x) dx = \frac{2(b-a)}{n} = \sum_{i=0}^{n-1} f\left(x_i + \frac{b-a}{2n}\right)$$

2.2.2. Kwadratura złożona trapezów

$$2 \int_{0}^{\infty} e^{-x^2} cos(x) dx = \frac{b-a}{n} = \sum_{i=0}^{n-1} (f(x_i) + f(x_{i+1}))$$

2.2.3. Kwadratura złożona Simpsona

$$2 \int_{0}^{\infty} e^{-x^2} cos(x) dx = \frac{2h}{3} = \sum_{i=1}^{\left[\frac{n}{2}\right]} \left(f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\right), h = \frac{b-a}{n}$$

2.2.4. Program do całkowania metodami złożonych kwadratur prostokątów, trapezów i Simpsona

Napisano program w języku Python z wykorzystaniem biblioteki NumPy do realizacji całkowania za pomocą kwadratur złożonych prostokątów, trapezów i Simpsona. Wykorzystano zmodyfikowany program przygotowany na poprzednich laboratoriach.

```
import math
import numpy as np
```

```python
import time

def f(x):
    if x == 0:
        return 0
    return math.cos(x) * math.exp(x ** -2)

def rectangle(a, b, n):
    time_start = time.time()

    dx = (b - a) / n
    suma = 0
    for i in range(n):
        x = a + i * dx
        suma += f(x)

    time_stop = time.time()

    return dx * suma, time_stop - time_start

def trapezoidal(a, b, n):
    time_start = time.time()

    dx = (b - a) / n
    suma = 0
    for i in range(n):
        x1 = a + i * dx
        x2 = a + (i + 1) * dx
        suma += (f(x1) + f(x2)) / 2

    time_stop = time.time()

    return dx * suma, time_stop - time_start

def simpson(a, b, n):
    time_start = time.time()

    dx = (b - a) / n
    suma = 0
    for i in range(n):
        x1 = a + i * dx
        x2 = a + (i + 1) * dx
        xm = (x1 + x2) / 2
        suma += (f(x1) + 4 * f(xm) + f(x2)) / 6

    time_stop = time.time()

    return dx * suma, time_stop - time_start

if __name__ == "__main__":
    a = 0
    b = 1000
    n = 1000
    rectangle_result, rectangle_time = rectangle(a, b, n)
    print("Metoda prostokątów: ", rectangle_result, rectangle_time)
    trapezoidal_result, trapezoidal_time = trapezoidal(a, b, n)
    print("Metoda trapezów: ", trapezoidal_result, trapezoidal_time)
    n = 100
    simpson_result, simpson_time = simpson(a, b, n)
    print("Metoda Simpsona: ", simpson_result, simpson_time)
```

2.2.5. Wyniki wydajnościowe

Wyniki wydajnościowe programu zaprezentowano w tabeli:

n	I_n	Błąd bezwględny	Czas
100	1.380397535648933	9.088e-6	4.100e-5
1000	1.380387897421244	5.596e-6	0.000442743
10000	1.380387953551231	4.934e-7	0.002820730

Tabela 1. Wyniki całkowania numerycznego z wykorzystaniem złożonej metody prostokątów

n	I_n	Błąd bezwględny	Czas
100	1.380388546201120	9.916e-8	6.699e-5
1000	1.380388495628143	4.858e-8	0.000621795
10000	1.380388464220912	1.717e-8	0.006979942

Tabela 2. Wyniki całkowania numerycznego z wykorzystaniem złożonej metody trapezów

n	I_n	Błąd bezwględny	Czas
100	1.380388447035643	7.499e-10	0.006979942
1000	1.380388447043075	6.797e-11	0.018733193
10000	1.380388447043142	9.747e-12	0.0899367212

Tabela 3. Wyniki całkowania numerycznego z wykorzystaniem złożonej metody Simpsona

2.2.6. Wnioski

Tak jak w poprzednim laboratorium pokazano, że kwadraturą złożoną prostokątów, trapezów i Simpsona można z dobrą dokładnością obliczyć wartość całki oznaczonej wykorzystując ideę „całki jako pola figury pod wykresem” (całka Riemanna).

Również tak jak poprzednio najdokładniejsza okazała się metoda Simpsona.

2.3. (b) Całkowanie adaptacyjne

2.3.1. Idea całkowania adaptacyjnego

Całkowanie adaptacyjne to metoda numerycznego całkowania funkcji, która polega na dzieleniu przedziału całkowania na mniejsze części i dokonywaniu przybliżenia całki w każdym z tych podprzedziałów. W trakcie tego procesu wykorzystywane są różne techniki, takie jak reguła trapezów czy Simpsona, które pozwalają na przybliżenie wartości całki w danym przedziale z dużą dokładnością.

Jedną z zalet całkowania adaptacyjnego jest to, że automatycznie dostosowuje się do złożoności funkcji, z której obliczana jest całka. Dzięki temu, że każdy podprzedział jest analizowany oddzielnie, metoda ta jest w stanie dokładnie określić wartość całki nawet w przypadku bardzo skomplikowanych funkcji.

W praktyce, całkowanie adaptacyjne jest szeroko stosowane w różnych dziedzinach, takich jak nauki przyrodnicze, inżynieria czy finanse, gdzie konieczne jest szybkie i dokładne rozwiązywanie skomplikowanych problemów matematycznych.

2.3.2. Obliczenie całki metoda całkowania adaptacyjnego

Ogólny algorytm tej metody wygląda następująco:

1 Podziel przedział całkowania $[a, b]$ na równe części i oblicz wartości funkcji w ich środkach.
2 Oblicz wartość całki na podprzedziałach tych części.
3 Porównaj wyniki obliczeń na sąsiadujących podprzedziałach. Jeśli różnią się one więcej niż o zadany próg tolerancji, to podprzedział zostaje podzielony na dwa mniejsze i proces jest powtarzany rekurencyjnie dla każdego z nich.
4 Zsumuj wyniki obliczeń dla wszystkich podprzedziałów.

2.3.3. Program do obliczenia całki metodą całkowania adaptacyjnego

Napisano program w języku Python z wykorzystaniem biblioteki NumPy do realizacji całkowania za pomocą całkowania adaptacyjnego z wykorzystaniem metody Simpsona.

```
import numpy as np
import time

def f(x):
    return np.exp(-x ** 2) * np.cos(x)

def simpson(f, a, b):
    h = b - a
    middle = (a + b) / 2

    return h * (f(a) + 4 * f(middle) + f(b)) / 6

def adaptive_quadrature(f, a, b, epsilon):
```

```
    mid = (a + b) / 2
    diff = abs(simpson(f, a, b) - simpson(f, a, mid) - simpson(f, mid, b))

    if diff < 15 * epsilon:
        return simpson(f, a, mid) + simpson(f, mid, b)
    return adaptive_quadrature(f, a, mid, epsilon / 2) +
adaptive_quadrature(f, mid, b, epsilon / 2)

if __name__ == "__main__":
    a = 0
    b = 10000
    epsilon = 1e-6
    time_start = time.time()
    print("Wynik: ", adaptive_quadrature(f, a, b, epsilon))
    time_end = time.time()
    print("Czas: ", time_end - time_start)
```

2.3.4. Wyniki Wydajnościowe

Wyniki wydajnościowe programu zaprezentowano w tabeli:

n	I_n	Błąd bezwzględny	Czas
100	1.380388447035642	7.500e-10	0.002191066741
1000	1.380388447043078	6.797e-11	0.004151105880
10000	1.380388447043142	9.747e-12	0.006895780563

Tabela 4. Wyniki całkowania numerycznego z wykorzystaniem złożonej metody Simpsona

2.3.5. Wnioski

Wnioski płynące z wykorzystania całkowania adaptacyjnego:

1. Zwiększenie dokładności wyniku: całkowanie adaptacyjne umożliwia uzyskanie bardziej dokładnych wyników niż tradycyjne metody numeryczne, takie jak kwadratura prostokątów lub metoda trapezów, ponieważ dostosowuje liczbę punktów obliczeniowych do złożoności funkcji.

2. Oszczędność czasu i zasobów: całkowanie adaptacyjne może zaoszczędzić czas i zasoby obliczeniowe, ponieważ algorytm zatrzymuje obliczenia, gdy uzyska wystarczającą dokładność, co oznacza, że nie musi wykonywać niepotrzebnych obliczeń.

3. Zastosowanie w wielu dziedzinach: całkowanie adaptacyjne jest szeroko stosowane w różnych dziedzinach, takich jak nauki przyrodnicze, inżynieria, ekonomia, finanse i wiele innych. Może być używane do obliczania całek jednowymiarowych i wielowymiarowych.

4. Potrzeba doświadczenia i oceny: całkowanie adaptacyjne wymaga doświadczenia w doborze odpowiednich parametrów i oceny błędu. Musi również zostać skonfigurowany w sposób odpowiedni dla konkretnej funkcji, aby uzyskać dokładne wyniki.

5. Istnieją różne algorytmy: istnieje wiele różnych algorytmów całkowani adaptacyjnego, takich jak algorytm Simpsona, algorytm Gaussa-Kronroda i wiele innych. Każdy z nich ma swoje zalety i wady i należy wybrać odpowiedni algorytm w zależności od potrzeb.

Metoda ta okazała się być bardzo dokładna i ma bardziej ogólne zastosowanie, ponieważ daje zadowalające wyniki także dla funkcji, których całkowanie innymi metodami daje mierne efekty.

2.4. (c) Kwadratura Gaussa-Hermite'a

2.4.1. Idea kwadratury Gaussa-Hermite'a do całkowania

Kwadratura Gaussa-Hermita jest stosowana do całkowania funkcji postaci:

$$w(x) \cdot g(x)$$

na przedziale $(-\infty, +\infty)$,

gdzie:

$$w(x) = exp(-x^2)$$

to waga kwadratury Gaussa.

W naszym przypadku:

$$g(x) = cos(x),$$

a cała funkcja całkowana jest postaci:

$$f(x) = g(x) \cdot e^{-x^2} = cos(x) \cdot e^{-x^2},$$

dodatkowo nasza całka obliczana jest na przedziale $(-\infty, +\infty)$, zatem możemy zastosować kwadraturę Gaussa-Hermite'a.

2.4.2. Obliczenie całki kwadraturą Gaussa-Hermite'a

Kwadratura Gaussa-Hermita jest jedną z metod numerycznych całkowania numerycznego funkcji jednej zmiennej. Polega na przybliżeniu wartości całki za pomocą węzłów i wag wyznaczonych na podstawie wielomianów ortogonalnych Hermita:

$$I(f) = \int_{-\infty}^{+\infty} e^{-x^2} \cdot f(x)dx \approx \sum_{i=1}^{n} w_i f(t_i),$$

gdzie: t_i to pierwiastki n-tego stopnia wielomianu Hermite'a.

Kwadratura Gaussa-Hermita jest stosowana do całkowania funkcji postaci:

$$w(x) \cdot g(x)$$

na przedziale $(-\infty, +\infty)$,

gdzie:

$$w(x) = exp(-x^2)$$

to waga kwadratury Gaussa.

W naszym przypadku:

$$g(x) = cos(x),$$

a cała funkcja całkowana jest postaci:

$$f(x) = g(x) \cdot e^{-x^2} = cos(x) \cdot e^{-x^2},$$

dodatkowo nasza całka obliczana jest na przedziale $(-\infty, +\infty)$, zatem możemy zastosować kwadraturę Gaussa-Hermite'a.

Wielomiany Hermite'a zdefiniowane są rekurencyjnie:

$$H_0(x) = 1,$$

$$H_1(x) = 2x,$$

$$H_{n+1}(x) = 2xH_n(x) - 2xH_{n-1}(x).$$

Kilka pierwszych wielomianów Hermite'a:

- $H_0 = 1$,
- $H_1 = 2x$,
- $H_2 = 4x^2 - 2$,
- $H_3 = 8x^3 - 12x$,
- $H_4 = 16x^4 - 48x^2 + 12$.

Wagi w_i wyrażają się zatem wzorem:

$$w_i = \frac{2^{n-1} \cdot n!}{n^2 \cdot [H_{n-1}(x_i)]^2} \cdot \sqrt{\pi},$$

gdzie:

- n – liczba węzłów,
- x_i – pierwiastki wielomianu Hermite'a stopnia n,
- H_n – wielomian Hermite'a stopnia n.

Obliczamy pierwiastki wielomianu H_4:

- $x_1 = -\frac{\sqrt{3+\sqrt{6}}}{2} = -1{,}650680;$
- $x_2 = -\frac{\sqrt{-3+\sqrt{6}}}{2} = -0{,}524647;$
- $x_3 = \frac{\sqrt{-3+\sqrt{6}}}{2} = 0{,}524647;$
- $x_4 = \frac{\sqrt{3+\sqrt{6}}}{2} = 1{,}650680;$

oraz liczmy wagi z wcześniejszego wzoru:

- $w_1 = 0{,}081313;$
- $w_2 = 0{,}804914;$
- $w_3 = 0{,}804914;$
- $w_4 = 0{,}081313.$

Ostatecznie otrzymujemy wynik:

$$\int_{-\infty}^{+\infty} e^{-x^2} \cdot cos(x) = 1{,}3803297572.$$

2.4.1. Program obliczenia całki metodą Gaussa-Hermite'a

Napisano program w języku Python z wykorzystaniem biblioteki NumPy do realizacji całkowania za pomocą kwadratury Gaussa-Hermite'a:

```
from math import cos
import time
import numpy as np

def f(x):
    return cos(x)

def gauss_hermite_integration(n, test=True):
    x, w = np.polynomial.hermite.hermgauss(n)
    if not test:
        print("weights:", *w)
        print("roots:", *x)

    return sum([w[i] * f(x[i]) for i in range(n)])

if __name__ == "__main__":
    for n in [4, 8, 16, 32, 64]:
        start_time = time.time()
        result = gauss_hermite_integration(n, True)
        end_time = time.time()
        print("n =", n, "\n", "result =", result, "\n", "time =", end_time
            - start_time, "seconds")
```

2.4.2. Wyniki wydajnościowe

Wykorzystano liczby typy $numpy.float64$ z biblioteki NumPy dla większej precyzji obliczeń.

n	Wynik	Czas [seksundy]
4	1.3803297571612558	0.0016548633575439453
8	1.3803884470313008	0.0004849433898925781
16	1.3803884470431427	0.0006580352783203125
32	1.3803884470431436	0.0011541843414306641
64	1.3803884470431431	0.0019657611846923833

Tabela 5. Wyniki wydajnościowe dla całkowania kwadraturą Gaussa-Hermite'a

2.4.3. Wnioski

Kwadratura Gaussa-Hermita jest szczególnie skuteczna w całkowaniu funkcji gładkich, które maleją szybko w nieskończoności.

3. Bibliografia

1. Wykłady dr inż. Katarzyny Rycerz z przedmiotu *Metody obliczeniowe w nauce i technice* na czwartym semestrze kierunku Informatyka w AGH w Krakowie
2. Wykresy kreślono za pomocą internetowego programu GeoGebra: https://www.geogebra.org/calculator
3. Obliczenia wykonywano za pomocą internetowego programu WolframAlpha: https://www.wolframalpha.com/ oraz programu Microsoft Excel: https://www.microsoft.com/pl-pl/microsoft-365/excel
4. Programy napisane zostały w języku Python w wersji 3.11: https://www.python.org/
5. Wykorzystano bibliotekę NumPy dla języka Python w wersji 1.24: https://numpy.org/doc/stable/index.html
6. Wykorzystano bibliotekę SciPy dla języka Python w wersji 1.10.1: https://scipy.org/
7. Kalkulator całek – *WolframAlpha Online Integral Calculator:* https://www.wolframalpha.com/calculators/integral-calculator/
8. https://en.wikipedia.org/wiki/Numerical_integration
9. https://en.wikipedia.org/wiki/Simpson%27s_rule
10. https://en.wikipedia.org/wiki/Riemann_sum
11. https://en.wikipedia.org/wiki/Trapezoidal_rule
12. https://pl.wikipedia.org/wiki/Ca%C5%82ka_Gaussa
13. https://pl.wikipedia.org/wiki/Kwadratury_Gaussa
14. https://pl.wikipedia.org/wiki/Wielomiany_Hermite%E2%80%99a
15. https://numpy.org/doc/stable/reference/generated/numpy.polynomial.hermite.hermgauss.html