Laboratorium 6

Całkowanie numeryczne II

Bartłomiej Szubiak

16.04.2024

Zadania

Zad 1

Obliczyć przybliżoną wartość całki:

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

a) przy pomocy złożonych kwadratur (prostokątów, trapezów, Simpsona),

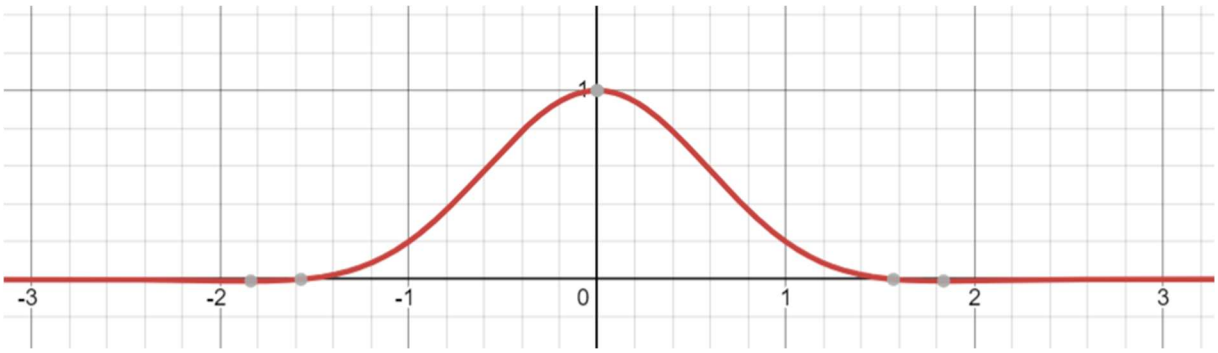
b) przy pomocy całkowania adaptacyjnego,

c) przy pomocy kwadratury Gaussa-Hermite'a, obliczając wartości węzłów i wag.

Porównać wydajność dla zadanej dokładności.

Zadana całka jest niemożliwa do obliczenia analitycznie, nie istnieje jej funkcja pierwotna.

Oby obliczyć wartość numerycznie musimy zawęzić przedział całkowania. Zauważmy że funkcja maleje (ma coraz mniejszą wartość) dość szybko im dalej znajdujemy się od punktu x=0.



Rysunek 1: Wykres funkcji $e^{-x^2}cos(x)$

Zauważmy, że człon ‘cos x’ nie wiele wpływa na trygonometryczny wygląd funkcji
tz. funkcja $e^{-x^2}cos(x)$ jest dość podobna do funkcji e^{-x^2}.

Zauważmy, że w punktach x=-10 , x=10 funkcja przyjmuje już wartość bardzo bliską 0, zawęszmy więc zakres całkowania z $[-\infty, +\infty]$ do $[-10,10]$. Całka z zakresu $R\backslash[-10,10]$ jest bardzo bliska 0, bo funkcja przyjmuje wartości bliskie 0.

a) z użyciem metody prostokątów, trapezów, Simpsona

Przedział $[-10,10]$ został podzielony na n podprzedziałów $n \epsilon \{ 10 , 100 , 1000 \}$:

Wyniki dla n= 10

Metoda prostokątów: 0.7945757422811989

Metoda trapezów: 1.9695117250368583

Metoda Simpsona: 2.6463409538701743

Wyniki dla n= 100

Metoda prostokątów: 1.380388447043143

Metoda trapezów: 1.380388447043143

Metoda Simpsona: 1.3803884470431427

Wyniki dla n= 1000

Metoda prostokątów: 1.3803884470431416

Metoda trapezów: 1.3803884470431427

Metoda Simpsona: 1.3803884470431427

Użyty kod python:

```
import numpy as np

# Definicja funkcji, którą całkujemy
def f(x):
    return np.exp(-x**2) * np.cos(x)

# Metoda prostokątów
def rectangle_method(f, a, b, n):
    h = (b - a) / n
    result = 0
    for i in range(n):
        x = a + i * h + h / 2  # środek przedziału
        result += f(x)
    result *= h
    return result

# Metoda trapezów
def trapezoidal_method(f, a, b, n):
    h = (b - a) / n
    result = 0.5 * (f(a) + f(b))
    for i in range(1, n):
        result += f(a + i * h)
    result *= h
    return result

# Metoda Simpsona
def simpsons_method(f, a, b, n):
    h = (b - a) / n
    result = f(a) + f(b)
    for i in range(1, n):
        if i % 2 == 0:
            result += 2 * f(a + i * h)
        else:
            result += 4 * f(a + i * h)
    result *= h / 3
    return result

# Wyniki dla różnych metod
for n in [10, 100, 1000]:
    result_rectangle = rectangle_method(f, -10, 10, n)
    result_trapezoidal = trapezoidal_method(f, -10, 10, n)
    result_simpsons = simpsons_method(f, -10, 10, n)

    print("Wyniki dla n=", n)
    print("Metoda prostokątów:", result_rectangle)
    print("Metoda trapezów:", result_trapezoidal)
    print("Metoda Simpsona:", result_simpsons)
    print()
```

Wnioski:
Jak można zauważyć im więcej przedziałów tym większa dokładność i poprawność wyniku, dla n=10 wyniki są dość odstające, a dla n=1000 są dużo bardziej dokładne i zbliżone do siebie z różnych metod kosztem mocy obliczeniowej

b) przy pomocy całkowania adaptacyjnego:

Całkowanie adaptacyjne jest techniką numerycznego obliczania całek, która adaptuje się do kształtu funkcji, aby zapewnić dokładność przy minimalnym koszcie obliczeniowym.

Zastosuję metoda Simpsona z podziałem przedziału na mniejsze podprzedziały, a następnie stosowania metody Simpsona w każdym z tych podprzedziałów.

Dla epsilon (dokładności) = 10^{-6}

Wynik (adaptacyjne): 1.3803883995741228

Użyty kod python:

```
def adaptive_simpsons(f, a, b, eps, whole_area=None):
    if whole_area is None:
        whole_area = (b - a) * (f(a) + 4 * f((a + b) / 2) + f(b)) / 6

    c = (a + b) / 2
    left_area = (c - a) * (f(a) + 4 * f((a + c) / 2) + f(c)) / 6
    right_area = (b - c) * (f(c) + 4 * f((c + b) / 2) + f(b)) / 6
    approx_area = left_area + right_area

    if abs(whole_area - approx_area) <= 15 * eps:
        return approx_area + (approx_area - whole_area) / 15

    return adaptive_simpsons(f, a, c, eps / 2, left_area) + adaptive_simpsons(f, c, b, eps / 2, right_area)

# Definicja funkcji, którą całkujemy
def f(x):
    return np.exp(-x**2) * np.cos(x)

# Obliczenie całki adaptacyjnie
result_adaptive = adaptive_simpsons(f, -10, 10, 1e-6)

print("Wynik (adaptacyjne):", result_adaptive)
```

Wnioski:
W tej implementacji możemy narzucić odgórnie dokładność, poprzez to ze dzielimy przedział całkowania rekurencyjnie na dwie części zyskujemy na mocy obliczeniowej na niektórych przedziałach ale za to zużywamy jej więcej na przedziałach gdzie funkcja gwałtownie zmienia wartość, zachowując dokładność

c) kwadratura Gaussa-Hermite'a:
Metoda Gaussa-Hermite'a jest techniką numerycznego całkowania używaną do obliczania całek funkcji wagowej e^{-x^2} mnożonej przez funkcję testową w przedziale od minus do plus nieskończoności, postaci:

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

gdzie t_i to pierwiastki n-tego wielomianu Hermite'a.
Wielomiany Hermite'a są postaci:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x),$$

gdzie:

$H_0(x) = 1,$
$H_1(x) = 2x.$

Siedem pierwszych wielomianów Hermite'a:

$$H_0 = 1$$
$$H_1 = 2x$$
$$H_2 = 4x^2 - 2$$
$$H_3 = 8x^3 - 12x$$
$$H_4 = 16x^4 - 48x^2 + 12$$
$$H_5 = 32x^5 - 160x^3 + 120x$$
$$H_6 = 64x^6 - 480x^4 + 720x^2 - 120$$
$$H_7 = 128x^7 - 1344x^5 + 3360x^3 - 1680x.$$

Wagi kwadratury Gaussa Hermite'a:

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

Wyniki z użyciem wielomianu stopnia n, $n \epsilon [2,3,4,5,6,7]$:

Wynik całki dla n = 2 : 1.347498463716813
Wynik całki dla n = 3 : 1.3820330713880475
Wynik całki dla n = 4 : 1.380329757161256
Wynik całki dla n = 5 : 1.3803900759356564
Wynik całki dla n = 6 : 1.3803884100507338
Wynik całki dla n = 7 : 1.3803884477540787

Użyty kod python:

```
import numpy as np
import math

hermit_coefficients = [
   [1],
   [2, 0],
   [4, 0, -2],
   [8, 0, -12, 0],
   [16, 0, -48, 0, 12],
   [32, 0, -160, 0, 120, 0],
   [64, 0, -480, 0, 720, 0, -120],
   [128, 0, -1344, 0, 3360, 0, -1680, 0]
]

hermit_functions = [np.poly1d(coefficients) for coefficients in hermit_coefficients]
hermit_roots = [function.roots for function in hermit_functions]

def hermite_weight(n,x):
   return (np.sqrt(np.pi) * math.factorial(n) * (2 ** (n - 1))) /  ( (n**2) *
np.polyval(hermit_functions[n-1], x)**2 )

# Funkcja, którą całkujemy
def f(x):
   return np.cos(x)

def gauss_hermite_integration(f, n):
   # Obliczenie miejsc zerowych i wag wielomianów Hermite'a
   nodes = hermit_roots[n]
   weights = [hermite_weight(n, node) for node in nodes]

   # Obliczenie całki jako sumy iloczynów wag i wartości funkcji w miejscach zerowych
   integral = sum(weights[i] * f(nodes[i]) for i in range(n))

   return integral

#uzycie
for n in [2,3,4,5,6,7]:
   result = gauss_hermite_integration(f, n)
   print("Wynik całki dla n =", n, ":", result)
```

Wnioski:
Metoda całkowania Gaussa Hermite’a jest bardzo dobra do rozwiązania tego problemu, nie musimy zawężać dziedziny całkowania, dodatkowo funkcja do całkowania bardzo się upraszcza (do cos x) Znając wagi i miejsca zerowe wielomianów Hermite’a, do całkowania potrzeba mniejszej mocy obliczeniowej.

Podsumowanie:

Zebrane wyniki:

- metoda prostokątów, trapezów, Simpsona, zawężenie przedziału całkowania do [-10,10]

n	prostokątów	trapezów	Simpsona
10	0.794575742281198	1.969511725036858	2.6463409538701743
100	1.380388447043143	1.380388447043143	1.3803884470431427
1000	1.380388447043141	1.380388447043142	1.3803884470431427

Tabela 1: Zebrane wyniki dla metod prostokątów, trapezów, Simpsona

- metoda, dla epsilon (dokładności) = 10^{-6}

  Wynik: 1.3803883995741228

- kwadratura Gaussa-Hermite'a, z użyciem wielomianów stopnia n:

n	Wyniki:
2	1.347498463716813
3	1.382033071388047
4	1.380329757161256
5	1.380390075935656
6	1.380388410050733
7	1.380388447754078

Tabela 2: Zebrane wyniki dla metody Gaussa Hermite'a

Najlepszą z metod dla rozwiązania tego projektu jest użycie kwadratury Gaussa Hermite'a.

Bibliografia:

wykład dr inż. Katarzyna Rycerz
https://www.wolframalpha.com/calculators/integral-calculator/
https://home.agh.edu.pl/~funika/mownit/lab5/what_is_gauss.html
https://pl.wikipedia.org/wiki/Kwadratury_Gaussa
https://pl.wikipedia.org/wiki/Wielomiany_Hermite%E2%80%99a