Laboratorium 06

Całkowanie numeryczne II

Adam Biśta, 15.04.2023

1 Treść zadań

Zadania

1. Obliczyć przybliżoną wartość całki:

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

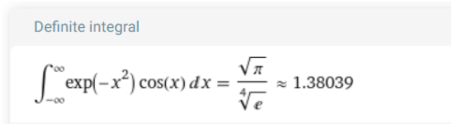
   (a) przy pomocy złożonych kwadratur (prostokątów, trapezów, Simpsona),

   (b) przy pomocy całkowania adaptacyjnego,

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

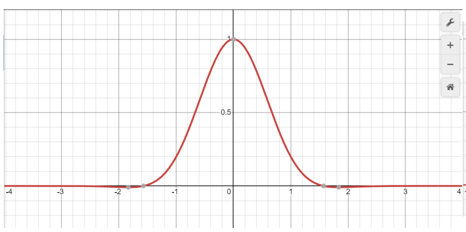
2 Rozwiązania zadań

Dokładny wynik obliczenia całki z wykorzystaniem programu wolfram:

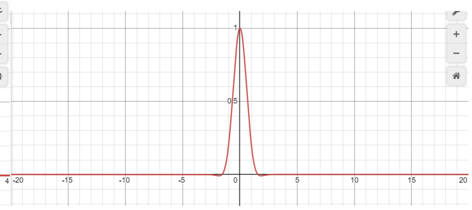


Rysunek 1

Dokładniejszy wynik (do 17 miejsc po przecinku): **1.38038844704314297**
Rozpatrzmy na wykresach funkcję $f(x) = e^{-x^2}\cos(x)$:



Rysunek 2: dla x: -4 do 4



Rysunek 3: dla x: -20 do 20

Widzimy, że funkcja ta przyjmuje znacznie większe wartości od zera w przedziale od ok. -1.5 do 1.5.

1. **kwadratury złożone**
   funkcja jest parzysta, tzn że f(-x) = f(x). Można więc przyjąć,że:

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

   Przyjąłem za ograniczenie górne $b = 10^5$
   Ograniczenie dolne $a = 0$
   liczba przedzialow tworzonych:

   - $n0 = 10^6$
   - $n1 = 5 \cdot 10^6$
   - $n2 = 7.5 \cdot 10^6$
   - $n3 = 10^7$

- Kwadratura złożona prostokątów:

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

- Kwadratura złożona trapezów:

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

- Kwadratura złożona Simpsona:

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

```
import numpy as np

from math import cos, e,pi,sqrt,exp

def f(x):
    return (e**(-x**2))*cos(x)


def rectangle(a, b, n):
    h = (b - a) / n
    args = np.linspace(a, b - h, n) + h/2
    res = 2 * sum(map(lambda x: f(x)*h,args))
    ans = [res,abs(real_ans - res)]
    return ans

def trapeze(a, b, n):
    h = (b - a) / n
    args = np.linspace(a, b, n+1)
    res =2* sum((h/2)*(f(args[i]) + f(args[i - 1])) for i in
    ↪  range(1,n+1))
    return [res,abs(real_ans-res)]

def simpson(a, b, n):
    if n % 2 != 0:
        n += 1 #aby suma ponizej sie zgadzala
    h = (b - a) / n
    args = np.linspace(a, b, n+1)
    res = ((2 * h) / 3) * sum(f(args[2 * i - 2]) + 4 *
    ↪  f(args[2 * i - 1]) + f(args[2 * i]) for i in
    ↪  range(1, (n // 2) + 1))
```

```
    return [res,abs(real_ans-res)]


if __name__ == "__main__":
    a = 0
    b = 10000
    N = [10**6,5*10**6,int(7.5*10**6),10**7]
    for n in N:
        print("n=",n)
        real_ans = sqrt(pi) / exp(1/4)
        print("Metoda prostokatow ")
        ans = rectangle(a, b,n)
        print("wynik ",ans[0])
        print("blad bezwzgledny ",ans[1])
        print("Metoda trapezow ")
        ans = trapeze(a, b,n)
        print("wynik ",ans[0])
        print("blad bezwzgledny ",ans[1])
        print("Metoda Simpsona ")
        ans = simpson(a, b,n)
        print("wynik ",ans[0])
        print("blad bezwzgledny ",ans[1])
```

n= **1000000**
Metoda prostokatow
wynik **1.380388447043143**
blad bezwzgledny **0.0**
Metoda trapezow
wynik **1.3803884470431436**
blad bezwzgledny **6.661338147750939e-16**
Metoda Simpsona
wynik **1.380388447043142**
blad bezwzgledny **8.881784197001252e-16**
n= **5000000**
Metoda prostokatow
wynik **1.3803884470431371**
blad bezwzgledny **5.773159728050814e-15**
Metoda trapezow
wynik **1.3803884470431378**
blad bezwzgledny **5.10702591327572e-15**
Metoda Simpsona
wynik **1.3803884470431382**
blad bezwzgledny **4.6629367034256575e-15**
n= **7500000**
Metoda prostokatow

wynik **1.380388447043143**
blad bezwzgledny **0.0**
Metoda trapezow
wynik **1.3803884470431371**
blad bezwzgledny **5.773159728050814e-15**
Metoda Simpsona
wynik **1.3803884470431385**
blad bezwzgledny **4.440892098500626e-15**
n= **10000000**
Metoda prostokatow
wynik **1.3803884470431373**
blad bezwzgledny **5.551115123125783e-15**
Metoda trapezow
wynik **1.380388447043131**
blad bezwzgledny **1.199040866595169e-14**
Metoda Simpsona
wynik **1.3803884470431418**
blad bezwzgledny **1.1102230246251565e-15**

2. **Całkowanie adaptacyjne**
   Wykorzystamy adaptacyjną kwadraturę Simpsona.

```
def simpson(f, a, b):
    h = b - a
    middle = (a + b) / 2
    return h * (f(a) + 4 * f(middle) + f(b)) / 6
```

   Tworzymy wzór rekurencyjny

```
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)
```

   Funkcja "adaptive quadrature"oblicza przybliżoną wartość całki z funkcji f na przedziale [a, b] z dokładnością do epsilon używając adaptacyjnej metody Simpsona. Oto jak działa krok po kroku:

   (a) Obliczamy środek przedziału mid jako średnią wartość a i b:

$$mid = \frac{a+b}{2}$$

(b) Obliczamy różnicę $diff$ jako wartość bezwzględną z różnicy między kwadraturą Simpsona na całym przedziale $[a, b]$ a sumą kwadratur Simpsona na dwóch podprzedziałach $[a, mid]$ i $[mid, b]$:

$$diff = |simpson(f, a, b) - (simpson(f, a, mid) + simpson(f, mid, b))|$$

(c) Sprawdzamy, czy różnica $diff$ jest mniejsza od 15 razy $epsilon$. Jeśli tak, to zwracamy sumę kwadratur Simpsona na podprzedziałach $[a, mid]$ i $[mid, b]$ jako przybliżoną wartość całki. Wybór wartości 15 w nierówności $diff < 15 \cdot epsilon$ pochodzi z analizy błędu metody adaptacyjnej kwadratury Simpsona. Wybranie tej wartości pozwala na uzyskanie odpowiedniej granicy błędu w przypadku adaptacyjnego całkowania.

(d) Jeśli różnica $diff$ jest większa niż 15 razy $epsilon$, rekurencyjnie wywołujemy funkcję $adaptivequadrature$ na dwóch podprzedziałach: $[a, mid]$ i $[mid, b]$, z połową tolerancji błędu ($\frac{\epsilon}{2}$), a następnie zwracamy sumę ich wyników jako przybliżoną wartość całki.

Dla tolerancji $\epsilon = 10^{-12}$ oraz $a = -1000$ i $b = 1000$ Wynik całki wynosi:

$$\int_a^b e^{-x^2} \cos(x)\, dx \approx 1.380388447043143$$

Aby stworzyć wykres wykorzystując podane całkowanie adaptacyjne, wykorzystamy dodatkową poniższą funkcję:

```python
def adaptive_quadrature_points(f, a, b, epsilon,
    points):
    s_q = simpson
    mid = (a + b) / 2
    diff = abs(s_q(f, a, b) - s_q(f, a, mid) - s_q(f,
        mid, b))
    if diff < 15 * epsilon:
        points.update([a, mid, b])
        return s_q(f, a, mid) + s_q(f, mid, b)
    return adaptive_quadrature_points(f, a, mid, epsilon
        / 2, points) + adaptive_quadrature_points(f,
        mid, b, epsilon / 2, points)
```

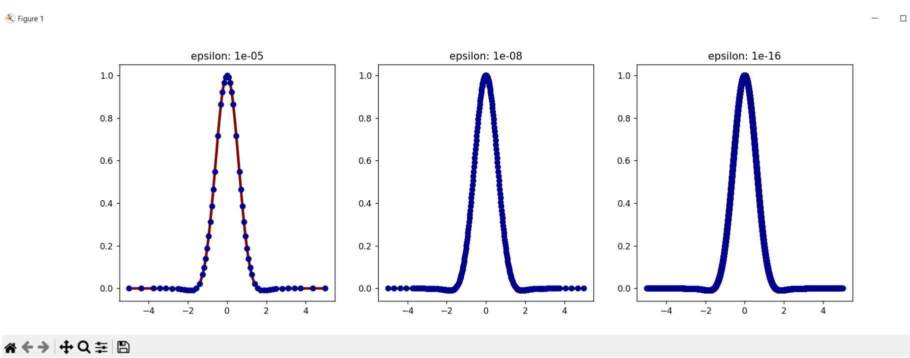
Funkcja $adaptivequadraturepoints$ oblicza przybliżoną wartość całki z funkcji f na przedziale [a, b] z dokładnością do ϵ oraz zbiera punkty adaptacyjne podczas procesu całkowania.

Kod generujący 3 wykresy wykorzystując całkowanie adaptacyjne z wyszczególnionymi punktami adaptacyjnymi:

```python
import numpy as np
import matplotlib.pyplot as plt

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)


def adaptive_quadrature_points(f, a, b, epsilon,
↪   points):
    s_q = simpson
    mid = (a + b) / 2
    diff = abs(s_q(f, a, b) - s_q(f, a, mid) - s_q(f,
    ↪   mid, b))
    if diff < 15 * epsilon:
        points.update([a, mid, b])
        return s_q(f, a, mid) + s_q(f, mid, b)
    return adaptive_quadrature_points(f, a, mid, epsilon
    ↪   / 2, points) + adaptive_quadrature_points(f,
    ↪   mid, b, epsilon / 2, points)

def plot_adaptive_points(f, a, b, epsilon, ax):
    points = set()
    adaptive_quadrature_points(f, a, b, epsilon, points)

    x_axis = np.linspace(a, b, 500)
    ax.plot(x_axis, f(x_axis), color='maroon',
    ↪   linewidth=3)
    points = np.array(list(points))
    ax.scatter(points, f(points), zorder=5,
    ↪   color="darkblue")
    ax.set_title(f'epsilon: {epsilon}')
```

```python
if __name__ == "__main__":
    a = -5
    b = 5
    _, ax = plt.subplots(1, 3, figsize=(18, 5))
    plot_adaptive_points(f, a, b, 1e-5, ax[0])
    plot_adaptive_points(f, a, b, 1e-8, ax[1])
    plot_adaptive_points(f, a, b, 1e-16, ax[2])
    plt.show()
```

Wynik wywołania programu:



3. **Kwadratura Gaussa-Hermit'a**
   Obliczamy całkę $\int_{-\infty}^{\infty} e^{-x^2} \cos x dx$ stosując metodę Gaussa-Hermite'a:

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

Węzły $H_n(x)$ są to pierwiastki wielomianu stopnia n.
Wagi natomiast możemy obliczyć wykorzystując wzór:

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

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

Aby obliczyć wagi i węzły, musimy wyznaczyć wielomiany Hermite'a:

$$H_0(x) = 1$$
$$H_1(x) = 2x$$
$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x)$$

Następnie obliczamy H_2, H_3 , H_4:

$$\begin{aligned}
H_2(x) &= 4x^2 - 2 \\
H_3(x) &= 8x^3 - 12x \\
H_4(x) &= 16x^4 - 48x^2 + 12
\end{aligned}$$

Pierwiastki wielomianu H_4:

$$\begin{aligned}
x_1 &= -\frac{\sqrt{3+\sqrt{6}}}{2} = -1.6507 \\
x_2 &= -\frac{\sqrt{-3+\sqrt{6}}}{2} = -0.5246 \\
x_3 &= \frac{\sqrt{-3+\sqrt{6}}}{2} = 0.5246 \\
x_4 &= \frac{\sqrt{3+\sqrt{6}}}{2} = 1.6507
\end{aligned}$$

Obliczamy teraz wagi na podstawie powyższych pierwiastków wykorzystując wzór na wagi podany wcześniej:
$w_1 = 0.0813128354 = \mathrm{w}_4$,
$\mathrm{w}_2 = 0.8049140900 = w_3$

Liczymy całkę:

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

$0.0813128354\cos(-1.6507)+$
$0.8049140900\cos(-0.5246)\ +$
$0.8049140900\cos(0.5246)+$
$0.0813128354\cos(1.6507) = 1.3803649357$

3 Wnioski

1. **zadanie 1:**
   Wyniki dla wszystkich trzech metod wykazują zbieżność, co sugeruje, że większa liczba iteracji n prowadzi do wyników bliższych rzeczywistej wartości całkowania. W przypadku każdej metody, wynik dla $n = 10^7$ jest niezwykle zbliżony do dokładnego wyniku całkowania. Błąd bezwzględny maleje w miarę wzrostu liczby iteracji n. Wszystkie trzy metody charakteryzują się prostotą implementacji.

2. **zadanie 2:**
   Im mniejsza tolerancja, tym błąd bezwzględny jest mniejszy. Gdy tolerancja wynosi 10^{-12}, błąd bezwzględny osiąga wartość bardzo bliską zera,

co wskazuje na wysoką dokładność uzyskanego wyniku. Metoda całkowania adaptacyjnego jest skomplikowana w implementacji, ale bardzo precyzyjna, gdyż automatycznie dostosowuje liczbę podziałów do potrzebnej dokładności.

3. **zadanie 3:**
   Określanie węzłów i wag dla danej liczby węzłów n oraz obliczanie wartości całki odbywa się automatycznie, przy użyciu wzorów rekurencyjnych na wielomiany Hermite'a. Kwadratura Gaussa-Hermita jest trudna w implementacji, jednak zapewnia bardzo precyzyjne wyniki nawet dla niewielkiej ilości podziałów.

4 Bibliografia

- Katarzyna Rycerz: "Wykład z przedmiotu Metody Obliczeniowe w Nauce i Technice"
- https://home.agh.edu.pl/ funika/mownit/lab5/calkowanie.pdf