Metody Obliczeniowe w Nauce i Technice
Natalia Bratek, 16.04.2024
Laboratorium 06

Całkowanie numeryczne II

1. Treści zadań

1.1 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.

2. Rozwiązanie

a) Złożone kwadratury (prostokątów, trapezów, Simpsona)

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

Metoda prostokątów:
Algorytm:

1. Przedział całkowania [a, b] dzielimy na n podprzedziałów równej długości

$$\Delta x = \frac{b-a}{n}$$

2. Wybór punktów próbnych:

   Dla każdego podprzedziału wybierany jest punkt, z którego wartość funkcji zostanie użyta do obliczenia pola prostokąta

3. Obliczanie sumy pól prostokątów:

Pole każdego prostokąta jest obliczane jako iloczyn wartości funkcji w wybranym punkcie i szerokości podprzedziału Δx.

W przypadku całek niewłaściwych z nieskończonymi granicami, musimy najpierw określić skończony przedział dla przybliżenia, ponieważ nie możemy bezpośrednio używać nieskończoności w obliczeniach numerycznych.
Rozpocznijmy od wybrania przedziału $[-a,a]$, gdzie a jest dużą liczbą, która ma służyć jako przybliżenie nieskończoności. Następnie podzielimy ten przedział na n równych części i użyjemy środków każdego z podprzedziałów do obliczenia wartości funkcji podcałkowej. Całka jest wtedy przybliżana przez sumę wartości funkcji w tych punktach środkowych, pomnożoną przez szerokość każdego z podprzedziałów.

```
import time
import numpy as np

a = 1000
n = 10000

real_value = 1.38039

def integrand(x):
    return np.exp(-x**2) * np.cos(x)
start_time = time.time()
x_values = np.linspace(-a, a, n+1)
dx = (x_values[-1] - x_values[0]) / n

mid_points = (x_values[:-1] + x_values[1:]) / 2
integrand_values = integrand(mid_points)

integral_approximation = np.sum(integrand_values) * dx
absolute_error = real_value - integral_approximation
end_time = time.time() - start_time
print(end_time)
print(integral_approximation)
print(absolute_error)
```

```
0.0010001659393310547
1.380388447043143
1.5529568571093222e-06
```

Rys.1 Metoda prostokątów

Metoda trapezów:

Metoda trapezów to kolejna technika numeryczna służąca do przybliżania wartości całek oznaczonych. W tej metodzie przedział całkowania jest dzielony na n równych części, ale zamiast używać wysokości prostokąta (jak w metodzie prostokątów), używamy wartości średniej z funkcji na początku i na końcu każdego przedziału, co przybliża obszar pod krzywą jako serię trapezów.

```
a = 1000
n = 10000
real_value = 1.38039

def integrand(x):
    return np.exp(-x**2) * np.cos(x)
start_time = time.time()
dx = (2 * a) / n
x_values = np.linspace(-a, a, n+1)

integrand_values = integrand(x_values)
integral_approximation_trapezoid = (integrand_values[0] + 2 * sum(integrand_values[1:-1]) + integrand_values[-1]) * dx / 2

absolute_error = real_value - integral_approximation_trapezoid
end_time = time.time() - start_time
print(end_time)
print(integral_approximation_trapezoid)
print(absolute_error)
```

```
0.006000041961669922
1.3803884470431427
1.5529568573313668e-06
```

Rys.2 Metoda trapezów

Metoda Simpsona:

Metoda Simpsona to metoda numeryczna do przybliżania wartości całek oznaczonych, która jest bardziej dokładna niż metoda trapezów dla tej samej liczby podziałów przedziału całkowania.

1. podzielenie przedziału całkowania na *n* równych części, gdzie *n* jest liczbą parzystą.
2. Dla każdego punktu *xi* na przedziale obliczana jest wartość funkcji podcałkowej, *f*(*xi*).
3. Wszystkie punkty dzielone są na grupy: pierwszy i ostatni punkt, punkty o parzystych indeksach oraz punkty o nieparzystych indeksach.
4. Obliczana jest suma wartości funkcji dla punktów o parzystych indeksach oraz sumę dla punktów o nieparzystych indeksach.
5. Całka przybliżana jest za pomocą następującej sumy:

$$\int_a^b f(x)\,dx \approx \frac{\Delta x}{3}\left[f(x_0) + 4\sum f(x_{\text{nieparzyste}}) + 2\sum f(x_{\text{parzyste}}) + f(x_n)\right]$$

```
a = 1000
n = 10000
start_time = time.time()
dx = (2 * a) / n

x_values = np.linspace(-a, a, n+1)
integrand_values = integrand(x_values)
sum_even = np.sum(integrand_values[2:-1:2])
sum_odd = np.sum(integrand_values[1::2])
integral_approximation_simpson = (dx/3) * (integrand_values[0] + 2*sum_even + 4*sum_odd + integrand_values[-1])
integral_approximation_simpson
absolute_error = real_value - integral_approximation_simpson
end_time = time.time() - start_time
print(end_time)
print(integral_approximation_simpson)
print(absolute_error)
```

```
0.002000093460083008
1.3803884470431527
1.5529568473393596e-06
```

Rys.3 Metoda Simpsona

b) Metoda całkowania adaptacyjnego

Dokładność dla metody adaptacyjnej 1e-6

Metoda całkowania adaptacyjnego to zaawansowany sposób numerycznego całkowania, który automatycznie dostosowuje gęstość punktów w miejscach, gdzie funkcja podcałkowa jest trudniejsza do przybliżenia. Dzięki temu możliwe jest osiągnięcie wyższej dokładności przy mniejszej liczbie obliczeń niż w standardowych metodach, które używają stałego podziału przedziału całkowania.
W metodzie adaptacyjnej proces podziału przedziału całkowania jest kontynuowany rekursywnie, aż do momentu, gdy osiągnięte przybliżenie całki na każdym podprzedziale będzie wystarczająco dokładne. Najczęściej stosuje się do tego kryterium błędu, na przykład błąd bezwzględny lub względny nie przekracza określonej wartości progowej. Algorytm:

1. Wybór początkowego podziału przedziału:
2. Obliczenie przybliżenia całki:
3. Ocena błędu.
4. Adaptacja podziału
5. Rekurencja:
6. Sumowanie wyników

```python
def adaptive_quadrature(f, a, b, tol, max_iter=50):

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

    def recursive_quadrature(f, a, b, tol, current_iter):
        c = (a + b) / 2
        one_simpson = simpson(f, a, b)
        two_simpsons = simpson(f, a, c) + simpson(f, c, b)
        if current_iter > max_iter:
            return two_simpsons
        elif abs(two_simpsons - one_simpson) < 15 * tol:
            return two_simpsons
        else:
            left = recursive_quadrature(f, a, c, tol/2, current_iter+1)
            right = recursive_quadrature(f, c, b, tol/2, current_iter+1)
            return left + right
    return recursive_quadrature(f, a, b, tol, 0)
start_time = time.time()
integral = adaptive_quadrature(integrand, -a, a, 1e-6)
absolute_error = real_value - integral
end_time = time.time() - start_time
print(end_time)
print(integral)
print(absolute_error)
```

```
0.017999887466430664
1.3803884464756802
1.5535243198527837e-06
```

Rys.4 Metoda całkowania adaptacyjnego

c) Kwadratura Gaussa – Hermite’a

Kwadratura Gaussa-Hermite'a jest zdefiniowana przez wzór:

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

Obliczanie węzłów i wag:

1) Wielomiany Hermite'a $H_n(x)$ są ortogonalne w odniesieniu do funkcji ważącej e^{-x^2} i mogą być wygenerowane rekurencyjnie:

$$H_0(x) = 1, \quad H_1(x) = 2x, \quad H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$$

2) Korzenie $H_n(x)$ są węzłami kwadratury Gaussa-Hermite'a. Znalezienie tych korzeni numerycznie można zrealizować za pomocą metody Newtona.
3) Wagi dla kwadratury Gaussa-Hermite'a są określone przez:

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

```python
def cos(x):
    from math import cos as math_cos
    return math_cos(x)
def gauss_hermite_quadrature(f, x, w):
    result = 0
    for xi, wi in zip(x, w):
        result += wi * f(xi)
    return result
def hermite_polynomial(n, x):
    if n == 0:
        return 1
    elif n == 1:
        return 2 * x
    else:
        return 2 * x * hermite_polynomial(n - 1, x) - 2 * (n - 1) * hermite_polynomial(n - 2, x)
def newton_method(f, df, x0, tol=1e-10, max_iter=100):
    x = x0
    for _ in range(max_iter):
        x_new = x - f(x) / df(x)
        if abs(x_new - x) < tol:
            return x_new
        x = x_new
    return x
def f(x):
    return hermite_polynomial(2, x)
def df(x):
    return 2 * 2 * hermite_polynomial(1, x)
start_time = time.time()
node1 = newton_method(f, df, 1.0)
node2 = newton_method(f, df, -1.0)
weight1 = (2 * 1 * (3.141592653589793)**0.5) / (4 * (hermite_polynomial(1, node1))**2)
weight2 = (2 * 1 * (3.141592653589793)**0.5) / (4 * (hermite_polynomial(1, node2))**2)
integral = gauss_hermite_quadrature(cos, nodes, weights)
absolute_error = real_value - integral
end_time = time.time() - start_time
print(end_time)
print(integral)
print(np.abs(absolute_error))
```

```
0.0009999275207519531
1.3820330713880475
0.0016430713880475345
```

Rys. 5 Kwadratura Gaussa-Hermite'a

Wartość obliczona za pomocą WolframAlpha wynosi w przybliżeniu 1.38039
Przyjęto: n = 10000
a = 1000

Metoda	Otrzymany wynik	Błąd	Czas
Prostokątów	1.3803884470431431	1.5529568568872776e-06	0.0010001659393310547
Trapezów	1.3803884470431427	1.5529568573313668e-06	0.006000041961669922
Simpsona	1.3803884470431527	1.5529568473393596e-06	0.002000093460083008
Całkowanie adaptacyjne	1.3803884464756802	1.5535243198527837e-06	0.017999887466430664
Gauss-Hermite	1.3820330713880475	0.0016430713880475345	0.0009999275207519531

Tab.1 Wyniki

Na podstawie danych w tabeli:

- Metody prostokątów, trapezów, Simpsona i całkowania adaptacyjnego dały bardzo zbliżone wyniki do siebie z błędem rzędu10^{-6}, co wskazuje na wysoką precyzję tych metod dla analizowanej funkcji
- Kwadratura Gaussa-Hermite'a miała znacznie większy błąd 10^{-3}w porównaniu z pozostałymi metodami, co sugeruje, że może nie być optymalna dla tej funkcji podcałkowej lub wymaga użycia większej liczby węzłów.
- Metoda prostokątów i trapezów wykazały najniższy czas wykonania, co jest oczekiwane, ponieważ są to najprostsze metody z najmniejszą liczbą obliczeń na podprzedział.
- Metoda Simpsona, będąca metodą bardziej złożoną, potrzebowała więcej czasu na wykonanie, ale nadal była stosunkowo szybka.
- Całkowanie adaptacyjne wykazało najdłuższy czas wykonania, co jest spowodowane koniecznością wielokrotnego obliczania całek na coraz mniejszych przedziałach i oceny błędu, aby dostosować dokładność.
- Kwadratura Gaussa-Hermite'a zajęła nieco mniej czasu niż całkowanie adaptacyjne, co może być spowodowane stosunkowo prostą formułą na wagi i węzły dla znanej funkcji ważącej
- Jeśli dokładność rzędu 10^{-6} jest wystarczająca, to proste metody numeryczne (prostokątów i trapezów) wydają się być najbardziej efektywne pod względem stosunku dokładności do czasu wykonania.
- Metoda Simpsona stanowi dobry kompromis między czasem wykonania a uzyskaną dokładnością.
- Całkowanie adaptacyjne może być preferowane w przypadkach, gdy funkcja podcałkowa jest skomplikowana lub gdy wymagana jest bardzo wysoka dokładność, pomimo większego czasu obliczeń.
- Kwadratura Gaussa-Hermite'a, mimo iż jest szybka, nie wydaje się być najlepszym wyborem dla tej funkcji podcałkowej, biorąc pod uwagę jej dokładność. Może to wynikać z niedopasowania metody do funkcji lub konieczności użycia większej liczby węzłów.

3. Bibliografia

- **https://www.wolframalpha.com/**
- https://pl.wikipedia.org/wiki/Kwadratury_Gaussa
- https://www.mimuw.edu.pl/~leszekp/dydaktyka/MO19L-g/adaptn.pdf
- https://home.agh.edu.pl/~funika/mownit/lab5/calkowanie.pdf