Metody Obliczeniowe w Nauce i Technice
Natalia Bratek, 14.04.2024
Laboratorium 05

# Całkowanie numeryczne

## 1. Treści zadań

1.1 Obliczyć $I = {}_0\int^1 1/(1+x)\,dx$ wg wzoru prostokątów, trapezów i wzoru Simpsona (zwykłego i złożonego n=3, 5). Porównać wyniki i błędy.
1.2 Obliczyć całkę $I = {}_{-1}\int^1 1/(1+x^2)\,dx$ korzystając z wielomianów ortogonalnych (np. Czebyszewa) dla n=8.
1.3 Obliczyć całkę ${}_0\int^1 1/(1+x^2)\,dx$ korzystając ze wzoru prostokątów dla h=0.1 oraz metody całkowania adaptacyjnego.
1.4 Metodą Gaussa obliczyć następującą całkę ${}_0\int^1 1/(x+3)\,dx$ dla n=4. Oszacować resztę kwadratury.

## 2. Rozwiązania

### 2.1 Obliczanie całki metodą prostokątów, trapezów, wzoru Simpsona

Dokładna wartość całki:

$$\int_0^1 \frac{1}{1+x}\,dx = [\ln(x+1)]_0^1 = \ln 2$$

Metoda prostokątów:
Jest używana do przybliżania wartości całki oznaczonej. Pozwala oszacować pole pod krzywą co jest równoznaczne z obliczeniem całki.

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 $\Delta x$. Suma tych pól daje przybliżenie całki:

   Przybliżona całka = $\Delta x \sum_{i=1}^{n} f(x_i)$

$x_i$ – wybrany punkt w i-tym podprzedziale

```
a, b = 0, 1
exact_integral = np.log(2)
n = [1, 3, 5]
for i in n:
    dx = (b - a) / i
    x_i = np.linspace(a, b - dx, i)
    sum_f = np.sum(f(x_i))
    approx_integral = sum_f * dx
    error = abs(exact_integral - approx_integral)
    print(approx_integral, error)
```

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.

$$Przybliżona\ całka = \frac{h}{2}(f(a) + 2\sum_{k=1}^{n-1} f(x_k) + f(b))$$

$$h = \frac{b-a}{n}$$

$$x_k = a + kh$$

```
a, b = 0, 1
exact_integral = np.log(2)
n = [1, 3, 5]
for i in n:
    dx = (b - a) / i
    x_i = np.linspace(a, b, i + 1)
    y_i = f(x_i)
    sum_f = np.sum(y_i[:-1] + y_i[1:])
    approx_integral = (dx / 2) * sum_f
    error = abs(exact_integral - approx_integral)
    print(f"n={i}: Approximation={approx_integral}, Error={error}")
```

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. Metoda Simpsona jest polega na podzieleniu przedziału całkowania [a, b] na n-1 równych podprzedziałów za pomocą n = 2k + 1, k ∈ N punktów, wyliczeniu wartości funkcji całkowanej tych punktach (ci) znajdujących się na krańcach ów podprzedziałów oraz w jego środku (jest to kwadratura trzypunktowa) a następnie policzeniu poniższej sumy.

$$Przybliżona\ całka = \sum_{i=3}^{n} \frac{h}{3} f(c_i) + 4f(c_{i-1}) + f(c_{i-2}))$$

$$h = \frac{b-a}{n}$$

```python
def simpson_rule(a, b, n):
    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    fx = f(x)
    integral = sum(h/3 * (fx[i] + 4 * fx[i - 1] + fx[i - 2]) for i in range(3, n + 1))
    return integral
a, b = 0, 1
exact_value = np.log(2)
for n in [3, 5]:
    approximation = simpson_rule(a, b, n)
    error = abs(exact_value - approximation)
    print(f"n={n}, Approximation={approximation}, Error={error}")
```

Rys.3 Metoda Simpsona

W wyniki przedstawiono otrzymane wyniki

| Metoda | n | Przybliżona całka | Błąd |
|---|---|---|---|
| prostokątów | 1 | 0.666666666667 | 0.0264805138933 |
| Prostokątów | 3 | 0.679365079365 | 0.0137821011949 |
| Prostokątów | 5 | 0.683852813853 | 0.00929436670713 |
| Trapezów | 1 | 0.75 | 0.056852819440054714 |
| Trapezów | 3 | 0.7 | 0.006852819440054669 |
| Trapezów | 5 | 0.6956349206349207 | 0.002487740074975431 |
| Simpsona | 3 | 0.694444444444 | 0.0012972638845 |
| Simpsona | 5 | 0.693253968254 | 0.000106787694023 |

Tabela 1. Wyniki

ln(2) = 0.69314718056

Wniosek:
Można zauważyć, że metoda Simpsona najlepiej przybliża całkę.

### 2.2 Obliczanie całki korzystając z wielomianów ortogonalnych

n=8

$$\int_{-1}^{1} \frac{1}{1+x^2} dx = [arctg(x)]_{-1}^{1} = \frac{\pi}{2} \approx 1{,}570796326794$$

Wzór na przybliżoną wartość całki przy użyciu kwadratury Gaussa-Czebyszewa typu pierwszego wygląda następująco:

$$I_n = \frac{\pi}{n} \sum_{i=1}^{n} f(x_i)$$

$I_n$ – przybliżona wartość całki

n – liczba węzłów
$\frac{\pi}{n}$ – waga każdego węzła
$x_i$ – i-ty węzeł
$f(x_i)$ – wartość funkcji podcałkowej

Dla każdego węzła waga jest stała i wynosi $\frac{\pi}{8}$

$$x_i = \cos\left(\frac{\pi(2i-1)}{2n}\right)$$

| n | wartości |
|---|---|
| 1 | 0.98078528 |
| 2 | 0.83146961 |
| 3 | 0.55557023 |
| 4 | 0.19509032 |
| 5 | -0.19509032 |
| 6 | -0.55557023 |
| 7 | -0.83146961 |
| 8 | -0.98078528 |

Tabela 2 Wartości węzłów

```
n = 8
def integrand(x):
    return 1 / (1 + x**2)
nodes = np.cos(np.pi * ( 2 *np.arange(1, n + 1) - 1) / (2 * n))
weight = np.pi / n
for i in range(n):
    print(f"Węzeł {i+1}: {nodes[i]}")
int = weights*np.sum(integrand(nodes))
print(int)
```

Rys. 4 kwadratura Gaussa - Czebyszewa

Otrzymujemy wynik: 1.5707963267948966
Błąd: 0.00000191424883

Wniosek: Kwadratura Gaussa- Czebyszewa daje bardzo dobre wyniki.

## 2.3 Obliczanie całki korzystając ze wzoru prostokątów i metody całkowania adaptacyjnego

Wzór prostokątów dla h = 0.1

$$\int_0^1 \frac{1}{1+x^2}dx = [arctg(x)]_0^1 = \frac{\pi}{4} \approx 0.785398163397$$

Jeżeli użyjemy *h*=0.1 jako szerokości każdego prostokąta, przedział całkowania [0,1] zostanie podzielony na 10 równych przedziałów.

Krok 1: Podział na przedziały

0, 0.1, …. 1.0

Krok 2: Wartości funkcji w środku przedziałów

Wartości funkcji w środku każdego przedziału, czyli w punktach: 0.05,0.15,0.25,…,0.95

Krok 3: Sumowanie wartości funkcji pomnożonych przez szerokość przedziału

Wzór na przybliżenie całki wynosi:

Przybliżona całka = $h \sum_{i=1}^{n} f(x_i)$

$x_i$ są punktami środkowymi przedziałów.

```
def f(x):
    return 1 / (1 + x**2)

def rectangle_method(f, a, b, h):
    x = np.arange(a + h/2, b, h)  # Punkty środkowe przedziałów
    return h * np.sum(f(x))
a = 0
b = 1
h = 0.1
rectangle_approx = rectangle_method(f, a, b, h)
print("Przybliżenie metodą prostokątów:", rectangle_approx)
```

Rys.5 Wzór prostokątów h= 0.1

Otrzymano: 0.7856064962502745
Błąd: 0.00020833285282606528

Metoda całkowania adaptacyjnego polega na dynamicznym dostosowywaniu przedziałów całkowania w zależności od zmienności funkcji, co pozwala na osiągnięcie wyższej dokładności w obszarach o wysokich gradientach funkcji podcałkowej. Jest to bardziej zaawansowana technika, która może być zaimplementowana np. przy użyciu metody Simpsona z adaptacją przedziałów.

1. Podział przedziału: Metoda zaczyna od podziału przedziału całkowania na mniejsze segmenty.
2. Ocena błędu: Dla każdego segmentu oceniany jest błąd przybliżenia całki, na przykład poprzez porównanie przybliżenia całki uzyskanego za pomocą

prostszych metod numerycznych (jak metoda trapezów lub Simpsona) z bardziej zaawansowanymi metodami.

3. Dostosowanie przedziałów: Segmenty, w których błąd przekracza założony próg, są dalej dzielone na mniejsze przedziały, a proces jest powtarzany rekurencyjnie.
4. Sumowanie wyników: Całkowita wartość całki jest sumą przybliżeń na wszystkich segmentach, przy czym na każdym etapie dokładność jest kontrolowana przez adaptacyjne dostosowywanie podziału przedziału.

```python
def f(x):
    return 1 / (1 + x**2)

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

def adaptive_simpson(f, a, b, epsilon, simpson_prev, depth=0):
    c = (a + b) / 2
    left_simpson = simpson(f, a, c)
    right_simpson = simpson(f, c, b)
    simpson_now = left_simpson + right_simpson

    if depth > 20:
        return simpson_now
    if abs(simpson_now - simpson_prev) > 15 * epsilon:
        left_result = adaptive_simpson(f, a, c, epsilon / 2, left_simpson, depth + 1)
        right_result = adaptive_simpson(f, c, b, epsilon / 2, right_simpson, depth + 1)
        return left_result + right_result
    else:
        return simpson_now

initial_estimate = simpson(f, 0, 1)
result = adaptive_simpson(f, 0, 1, 1e-6, initial_estimate)
print(result)
print((np.pi)/4 - result)
```

Rys.6 Metoda całkowania adaptacyjnego

Otrzymano: 0.7853978081111028

Błąd: 3.552863454547861e-07

Wniosek: Metoda adaptacyjna jest zalecana, gdy potrzebna jest wysoka dokładność i gdy funkcja podcałkowa jest skomplikowana. Metoda prostokątów może być użyteczna dla szybkich, przybliżonych obliczeń w mniej wymagających aplikacjach.

## 2.4 Obliczanie całkę i oszacować kwadraturę

n = 4

$\int_0^1 \frac{1}{3+x} dx$ = ln4 – ln3 ≈ 0.28768207245178

Aby zastosować wzory z przedziału [-1,1] w przedziale [a, b] należy dokonać transformacji liniowej zmiennej niezależnej:

$$t = \frac{a+b}{2} + \frac{b-a}{2}x$$

$$\int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} g(x)dx$$

$$g(x) = f\left(\frac{a+b}{2} + \frac{b-a}{2}x\right)$$

$$\int_a^b f(t)dt \approx S(f) = \frac{b-a}{2}\sum_{k=0}^{N} A_k f(t_k) \qquad t_k = \frac{a+b}{2} + \frac{b-a}{2}x_k$$

$$\int_0^1 f(t) \approx \frac{1-0}{2}\int_{-1}^{1} g(x)dx$$

$g(x)$ = f($\frac{1-0}{2}x + \frac{1+0}{2}$) $= f(\frac{1}{2}x + \frac{1}{2})$ = $\frac{1}{\frac{1}{2}x+\frac{7}{2}}$

S(f) = $\frac{1}{2}\sum_{k=0}^{4} w_k g(t_k)$

Używając kwadratury Gaussa-Legendre'a dla *n*=4

- Węzły (t): [−0.861136,−0.339981,0.339981,0.861136]
- Wagi (w): [0.347855,0.652145,0.652145,0.347855]

```python
import numpy as np

nodes = np.array([-0.861136311594053, -0.339981043584856, 0.339981043584856, 0.861136311594053])
weights = np.array([0.347854845137454, 0.652145154862546, 0.652145154862546, 0.347854845137454])

def transformed_function(t):
    return 1 / (0.5 * t + 3.5)

integral_approx = 1/2 * np.sum(weights * transformed_function(nodes))
print("Przybliżona wartość całki:", integral_approx)
```

Rys.7 Metoda Gaussa

Otrzymana wartość: 0.2876820721506314

Błąd: 0.000000000301149327687

Wniosek: Otrzymaliśmy bardzo dobre wyniki oraz przybliżenie. Kwadratura Gaussa jest wysoce efektywna.

Szacowanie reszty kwadratury:

Dokładność kwadratury Gaussa zależy od stopnia wielomianu funkcji podcałkowej. Kwadratura Gaussa-Legendre'a stopnia *n* jest dokładna dla wielomianów do stopnia 2*n*−1. W przypadku funkcji, która nie jest wielomianem, błąd (reszta kwadratury) jest trudniejszy do analitycznego oszacowania bez dodatkowych założeń o funkcji. Jednak, dla wielu praktycznych zastosowań kwadratura Gaussa zapewnia wysoką dokładność. Jeśli funkcja podcałkowa jest gładka i nie zawiera osobliwości w obrębie przedziału całkowania, błąd często maleje ekspotencjalnie zwiększając liczbę węzłów *n*. W praktyce, dla większości gładkich funkcji, błąd dla *n*=4 jest zazwyczaj bardzo mały.
Oszacowanie reszty kwadratury Gaussa-Legendre'a dla funkcji, która nie jest wielomianem, wymaga zazwyczaj obliczenia wyższych pochodnych funkcji. Wzór na resztę (błąd) dla kwadratury Gaussa-Legendre'a jest następujący:

$$R(f) = \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi)$$

gdzie:

- $R(f)$ to błąd kwadratury,
- $a, b$ to granice przedziału całkowania po transformacji,
- $n$ to liczba węzłów kwadratury,
- $f^{(2n)}(\xi)$ to $2n$-ta pochodna funkcji podcałkowej, ewaluowana w pewnym nieznanym punkcie $\xi$ z przedziału $[a, b]$.

Znalezienie szacunku maksymalnej wartości *k*-tej pochodnej na przedziale [0, 1] pozwoli nam na oszacowanie błędu. Maksymalna wartość tej pochodnej na przedziale [0, 1] będzie miała miejsce w punkcie *x*=0, ze względu na to, że mianownik jest wówczas najmniejszy. Aby więc oszacować resztę kwadratury dla *n*=4, musimy obliczyć 8-mą pochodną naszej funkcji podcałkowej i znaleźć jej wartość dla *x*=0. Następnie wstawiamy ją do powyższego wzoru.

## 3. Bibliografia


- https://home.agh.edu.pl/~dpawlus/pliki/matlab/MO_algorytmy.pdf
- https://pl.wikipedia.org/wiki/W%C4%99z%C5%82y_Czebyszewa
- https://www.mimuw.edu.pl/~leszekp/dydaktyka/MO19L-g/adaptn.pdf
- https://home.agh.edu.pl/~funika/mownit/lab5/what_is_gauss.html
- https://home.agh.edu.pl/~funika/mownit/lab5/calkowanie.pdf
- https://www.fuw.edu.pl/~jnareb/zajecia/int-gauss.pdf
- https://pl.wikipedia.org/wiki/Kwadratury_Gaussa