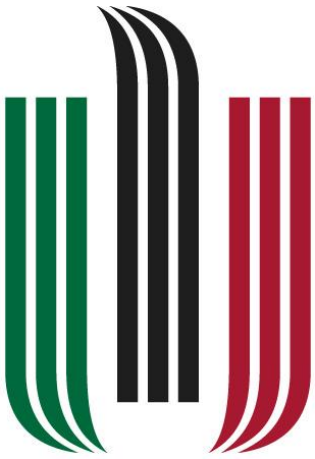
Laboratorium V

Całkowanie numeryczne

Dominik Marek

9 kwietnia 2024



AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Zadania

1. Obliczyć $I = \int_0^1 \frac{1}{1+x} dx$ wg wzoru prostokątów, trapezów i wzoru Simpsona (zwykłego i złożonego n=3, 5). Porównać wyniki i błędy.
2. Obliczyć całkę $I = \int_{-1}^1 \frac{1}{1+x^2} dx$ korzystając z wielomianów ortogonalnych (np. Czebyszewa) dla n=8.

Zadania domowe

1. Obliczyć całkę $I = \int_0^1 \frac{1}{1+x^2} dx$ korzystając ze wzoru prostokątów dla h=0.1 oraz metody całkowania adaptacyjnego.
2. Metodą Gaussa obliczyć następującą całkę $I = \int_0^1 \frac{1}{x+3} dx$ dla n=4. Oszacować resztę kwadratury.

2.Rozwiązania

2.1. Zadanie 1

Początkowo obliczam rzeczywistą wartość zadanej całki:

$$S_R(f) = \int_0^1 \frac{1}{1+x}\,dx = [ln|1+x|]_0^1 = ln\,2 - ln\,0 = ln\,2 \approx 0.69314718056$$

1. Metoda prostokątów

Polega ona na podzieleniu przedziału całkowania [a, b] na n równych podprzedziałów, wyliczeniu wartości funkcji całkowanej w punktach będących środkiem podprzedziałów a następnie obliczeniu następującej sumy:

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

W naszym zadaniu granice całkowania wynoszą odpowiednio $a = 0\ i\ b = 1$, zatem otrzymujemy następujące przybliżenie funkcji:

$$S(f) = \frac{1}{n}\sum_{i=1}^{n} f\left(x_i + \frac{1}{2n}\right)$$

Nasza całka jest więc równa sumie pól prostokątów o szerokości każdego podprzedziału i wysokości równej wartości funkcji całkowanej pośrodku tego przedziału.
Stosując powyższy wzór dla poszczególnej liczby przedziałów otrzymujemy poniższe zestawienie:

n	$\boldsymbol{S(f)}$	$\boldsymbol{\lvert S_R(f) - S(f)\rvert}$
1	0.6666666666666666	0.026480513893278657
3	0.6897546897546897	0.00339249080525561
5	0.6919078857159353	0.001239294844009975

Tablica 1: Porównanie wyników dla metody prostokątów

2. Metoda trapezów

Powyższa metoda polega na podzieleniu przedziału całkowania [a, b] na n-1 równych podprzedziałów za pomocą n punktów, wyliczeniu wartości funkcji całkowanej tych punktach znajdujących się na krańcach owych podprzedziałów odległych o h a następnie policzeniu poniższej sumy:

$$S(f) = \int_a^b f(x)\,dx = \frac{b-a}{2n}\sum_{i=2}^{n}[f(x_{i-1}) + f(x_i)]$$

Przyjmując żadne wyżej granice całowania otrzymujemy wzór poniższej postaci:

$$S(f) = \frac{1}{2n}\sum_{i=2}^{n}[f(x_{i-1}) + f(x_i)]$$

Stosując powyższy wzór dla poszczególnej liczby przedziałów otrzymujemy poniższe zestawienie:

n	$\boldsymbol{S(f)}$	$\boldsymbol{\lvert S_R(f) - S(f)\rvert}$
2	0.75	0.056852819440054714
3	0.7	0.006852819440054669
5	0.6956349206349207	0.00248774007497532

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

$$S(f) = \int_a^b f(x)\,dx = \frac{b-a}{3n}\sum_{i=3}^{n}[f(x_{i-2}) + 4f(x_{i-i}) + f(x_i)]$$

   Dla przyjętych granic całkowania otrzymujemy następujące przybliżenie funkcji:

$$S(f) = \frac{1}{3n}\sum_{i=3}^{n}[f(x_{i-2}) + 4f(x_{i-i}) + f(x_i)]$$

   Stosując powyższy wzór dla poszczególnej liczby przedziałów otrzymujemy poniższe zestawienie:

n	$S(f)$	$\lvert S_R(f) - S(f)\rvert$
3	0.6932539682539683	0.0001067876940229473
5	0.6931697931697931	0.00002612609847927345

Kwadratury zwykłe powyższymi metodami sprowadzają się do nie dzielenia przedziału nad podprzedziały. Są one tożsame z poniższymi kwadraturami złożonymi:

- Zwykła kwadratura prostokątów jest tożsama ze złożoną kwadraturą prostokątów dla n = 1
- Zwykła kwadratura trapezów jest tożsama ze złożoną kwadraturą trapezów dla n = 2
- Zwykła kwadratura Simpsona jest tożsama ze złożoną kwadraturą Simpsona dla n = 3, (dla k=1)

Powyższe wyniki zostały otrzymane po wykonaniu poniższego kodu napisanego w języku Python:

```
from typing import Any, Final
import numpy as np
from math import log
from numpy import ndarray, dtype

def fun(x:float|ndarray[Any, dtype[Any]]) -> float:
    return 1/(x+1)

def rectangle_method(r_min: int, r_max: int, n: int) -> float:
    width =  (r_max-r_min)/n
    integral_approx = 0
    pieces = np.linspace(r_min, r_max-width,n)
    pieces += width/2
    for x in pieces:
        integral_approx += fun(x)*width

    return integral_approx


def trapezoidal_method(r_min:int, r_max: int, n: int) -> float:
    pieces = np.linspace(r_min, r_max, n+1)
    return np.trapz(fun(pieces), pieces)


def simpson_method(r_min: int, r_max: int, n: int) -> float:
    width = (r_max-r_min)/(n+1)
    integral_approx = 0
    pieces = np.linspace(r_min, r_max, n+2)
```

```
    for i in range(1, (n+1)// 2 + 1):
        integral_approx += (fun(pieces[2*i-2]) + 4*fun(pieces[2*i-1]) +
fun(pieces[2*i]))
    return (integral_approx*width)/3


def main() -> None:
     MIN_RANGE: Final = 0
     MAX_RANGE: Final = 1
     num = [1, 3, 5]
     real_integral_value =  log(2)
     for n in num:
         print("Rectangle method")
         rectangle_method_res = rectangle_method(MIN_RANGE, MAX_RANGE, n)
         print(rectangle_method_res)
         print(abs(rectangle_method_res-real_integral_value))
         print("Trapezoidal method")
         trapezoidal_method_res = trapezoidal_method(MIN_RANGE, MAX_RANGE,
n)
         print(trapezoidal_method_res)
         print(abs(trapezoidal_method_res - real_integral_value))
         print("Simpson's method")
         simpson_method_res = simpson_method(MIN_RANGE, MAX_RANGE, n)
         print(simpson_method_res)
         print(abs(simpson_method_res - real_integral_value))

if __name__ == '__main__':
    main()
```

Wnioski:

Jak można się było spodziewać metoda Simpsona daje najlepsze oszacowanie, natomiast pomiędzy dwiema pozostałymi metodami, czyli metoda prostokątów oraz trapezów nie widać znaczących różnic w aproksymacji. Natomiast zauważalny jest wzrost dokładności aproksymacji wraz ze zwiększaniem liczby n.

2.2 Zadanie 2

Zaczynamy od wyznaczenia rzeczywistej wartości całki:

$$S_R(f) = \int_{-1}^{1} f(x)\,dx = \int_{-1}^{1} \frac{1}{1+x^2}dx = [arctan\,x]_0^1 = arctan\,1 - arctan(-1) = \frac{\pi}{2} \approx 1.570796326794$$

Dla N=8 aproksymujemy wartość całki w następujący sposób:

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=0}^{8} c_i f(x)\,, x\epsilon[-1,1]$$

Dzięki ortogonalności wielomianów Czebyszewa możemy wyznaczyć współczynniki c_i ze wzoru;

$$c_o = \frac{1}{\pi}\int_{-1}^{1}\frac{f(x)T_0(x)\,dx}{\sqrt{1-x^2}}$$

$$c_i = \frac{2}{\pi}\int_{-1}^{1}\frac{f(x)T_i(x)\,dx}{\sqrt{1-x^2}}\,, i \geq 1$$

Gdzie $T_i(x)$ to kolejne wielomiany Czebyszewa.

Wykorzystując powyższe informacje otrzymuje następujące przybliżenie :

$$S(f) \approx 1.570792527505668$$

Natomiast błąd bezwzględny wynosi:

$$|S_R(f) - S(f)| \approx 0.0000037992883$$

Powyższe wyniki otrzymałem posługując się poniższym programem napisanym w języku Python:

```
import  numpy as np
import scipy.integrate as spi
from math import pi


def integrate(fun, r_min:float, r_max: float) -> float:
    return spi.quad(fun, r_min, r_max)[0]


def chebyshev_polynomials(n):
    p = [np.poly1d([1]), np.poly1d([1, 0])]
    x = np.poly1d([1, 0])
    for i in range(2, n+1):
        p.append(2*x*p[i-1] - p[i-2])
    return p

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


def approx_fun(n: int):
    CH = chebyshev_polynomials(n)
    c = []
    fA = [np.poly1d([0])]
    for i in range(n+1):
        c.append(
            (1/pi if i == 0 else 2/pi)*spi.quad(
                (lambda x: f(x) * CH[i](x)/(1-x**2)**0.5), -1, 1
            )[0]
        )
        fA.append(c[i] * CH[i])
    return sum(fA)


def integration(n):
    integer = approx_fun(n).integ()
    return integer(1) - integer(-1)
```

```
def main() -> None:
    print(f'{integration(8)=}')
    print(f'{abs(integration(8) -pi/2) =}')


if __name__ == '__main__':
    main()
```

2.3 Zadanie domowe 1

Korzystając we wzorów wyprowadzonych z pierwszym zadaniu w celu rozwiązania tego zadania wystarczy jedynie zaktualizować funkcję dla której przeprowadzamy obliczenia oraz liczbę n, które w naszym przypadku wynoszą odpowiednio $f(x) = \frac{1}{1+x^2}$, n = 10.

Rozpoczynam od obliczenia rzeczywistej wartości całki:

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

Poniższa tabela przedstawia porównanie wyników dla poszczególnych metod;

Metoda	$S(f)$	$\lvert S_R(f) - S(f)\rvert$
Metoda prostokątów	0.7856064962502743	0.00020833285282606528
Metoda trapezów	0.7849814972267897	0.00041666617065860834
Metoda Simpsona	0.7853981534848038	0.00000000991264448302

Podobnie jak w zadaniu pierwszym posłużyłem się poniższym kodem w celu wyliczenia wartości funkcji:

```
from typing import Any, Final
import numpy as np
from math import pi
from numpy import ndarray, dtype

def fun(x:float|ndarray[Any, dtype[Any]]) -> float:
    return 1/(x**2+1)

def rectangle_method(r_min: int, r_max: int, n: int) -> float:
    width =  (r_max-r_min)/n
    integral_approx = 0
    pieces = np.linspace(r_min, r_max-width,n)
    pieces += width/2
    for x in pieces:
        integral_approx += fun(x)*width

    return integral_approx


def trapezoidal_method(r_min:int, r_max: int, n: int) -> float:
    pieces = np.linspace(r_min, r_max, n+1)
    return np.trapz(fun(pieces), pieces)
```

```
def simpson_method(r_min: int, r_max: int, n: int) -> float:
    width = (r_max-r_min)/n
    integral_approx = 0
    pieces = np.linspace(r_min, r_max, n+1)
    for i in range(1, n// 2 + 1):
        integral_approx += (fun(pieces[2*i-2]) + 4*fun(pieces[2*i-1]) +
fun(pieces[2*i]))
    return (integral_approx*width)/3



def main() -> None:
     MIN_RANGE: Final = 0
     MAX_RANGE: Final = 1
     n = 10
     real_integral_value =  pi/4
     print(f'{real_integral_value=}')
     print("Rectangle method")
     rectangle_method_res = rectangle_method(MIN_RANGE, MAX_RANGE, n)
     print(rectangle_method_res)
     print(abs((rectangle_method_res-
real_integral_value)/rectangle_method_res))
     print("Trapezoidal method")
     trapezoidal_method_res = trapezoidal_method(MIN_RANGE, MAX_RANGE, n)
     print(trapezoidal_method_res)
     print(abs((trapezoidal_method_res - real_integral_value) /
trapezoidal_method_res))
     print("Simpson's method")
     simpson_method_res = simpson_method(MIN_RANGE, MAX_RANGE, n)
     print(simpson_method_res)
     print(abs((simpson_method_res - real_integral_value) /
simpson_method_res))

if __name__ == '__main__':
    main()
```

Wnioski:

Podobnie jak w zadaniu 1 metoda Simpsona daje najlepsze oszacowanie, natomiast pomiędzy dwiema pozostałymi metodami, czyli metoda prostokątów oraz trapezów nie widać znaczących różnic w aproksymacji. Warto również nadmienić, że przy relatywnie małej liczbie przedziałów otrzymana wartość za pomocą całkowania numerycznego różni się o bardzo niewiele od wartości rzeczywistej.

2.2 Zadanie domowe 2

Ponownie zadanie zaczynamy od wilczenia rzeczywistej wartości całki ;

$$\int_0^1 f(x)\,dx = \int_0^1 \frac{1}{x+3}dx = [ln|x+3|]_0^1 = ln\,4 \approx\ 0.28768207245178$$

Musimy dokonać przekształcenia przedziału $(0, 1)$ na przedział $(-1, 1)$ i stąd otrzymujemy wzór na wartość aproksymowaną postaci:

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

Gdzie $g(x) = f\left(\frac{1-0}{2}\cdot x + \frac{1+0}{2}\right) = f\left(\frac{x+1}{2}\right)$

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

k	$\boldsymbol{w_k}$	$\boldsymbol{g(t_k)}$
1	0.6521451548625461	-0.3399810435848563
2	0.6521451548625461	0.3399810435848563
3	0.3478548451374538	-0.8611363115940526
4	0.3478548451374538	0.8611363115940526

Po wstawieniu wartości otrzymujemy: S(f) = 0.2876820721506314, zaś błąd bezwzględny równy:
$|S_R(f) - S(f)| = 0.000000000301149327697$

Wnioski:

Widzimy, że już dla n = 4 otrzymujemy świetne przybliżenie wartości całki.

3 Bibliografia

1. Włodzimierz Funika Materiały ze strony
2. Katarzyna Rycerz Wykład z przedmiotu Metody Obliczeniowe w Nauce i Technice
3. https://www.wolframalpha.com
4. https://pomax.github.io/bezierinfo/legendre-gauss.html
5. https://pomax.github.io/bezierinfo/legendre-gauss.html