Kwadratury

Joanna Klimek

Informatyka, WIET

18.04.2021r.

Zadanie 1.

Oblicz wartość całki

$$\int_{-1}^{1} \frac{2}{1+x^2}\,dx$$

korzystając ze wzorów prostokątów, trapezów i Simpsona.

Wykorzystując fakt, że

$$\int_{-1}^{1} \frac{2}{1+x^2}\,dx = \pi$$

dla każdej metody narysuj wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej, $n\ +\ 1$ (gdzie $n\ =\ \frac{1}{h}$, z krokiem h).

Metoda prostokątów:

Idea: Zadany przedział $[a, b]$ dzielimy na n równych przedziałów. Dla każdego punktu c dzielącego dany podprzedział na pół obliczamy wartość funkcji. Wartość całki przybliżamy sumą iloczynów długości podprzedziału i wyliczonej wartości funkcji w punkcie środkowym podprzedziału (pole powstałego prostokąta).

$$\int_a^b f(x)dx = \sum_{i=1}^{n} f(c_i) * \frac{b-a}{n}$$

Metoda trapezów:

Idea: Zadany przedział $[a, b]$ dzielimy na n równych przedziałów. Dla każdego punktu c dzielącego przedział na podprzedziały oraz w punktach końcowych obliczamy wartość funkcji. Wartość całki przybliżamy sumą iloczynów długości podprzedziału i sumy wyliczonych wartości funkcji w punktach końcowych podprzedziału, podzielonych przez 2 (pole powstałego trapezu prostokątnego).

$$\int_a^b f(x)dx = \sum_{i=2}^{n} \frac{(f(c_i) + f(c_{i-1})) * \frac{b-a}{n}}{2}$$

Metoda Simpsona:

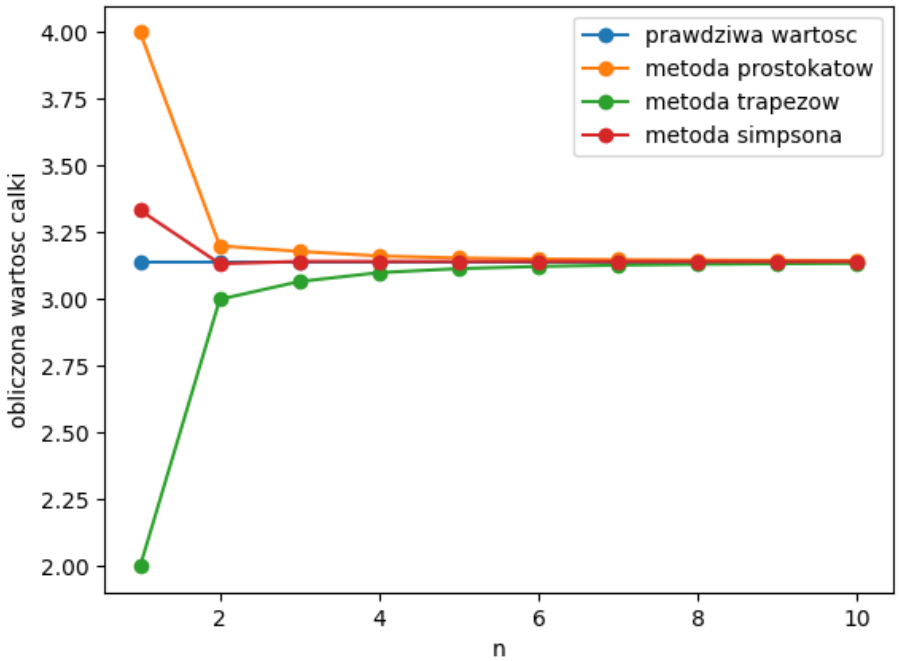
Idea: Zadany przedział $[a, b]$ dzielimy na n równych przedziałów. Dla każdego punktu c dzielącego cały przedział lub dzielącego dany podprzedział na pół obliczamy wartość funkcji. Wartość całki przybliżamy sumą iloczynów długości podprzedziału i sumy wyliczonych odpowiednich wartości funkcji w odpowiednich proporcjach, podzielonych przez 6, zgodnie z poniższym wzorem:

$$\int_a^b f(x)dx = \sum_{i=2}^{n} \frac{(f(c_i) + 4f\left(\frac{c_{i-1} + c_i}{2}\right) + f(c_{i-1})) * \frac{b-a}{n}}{6}$$

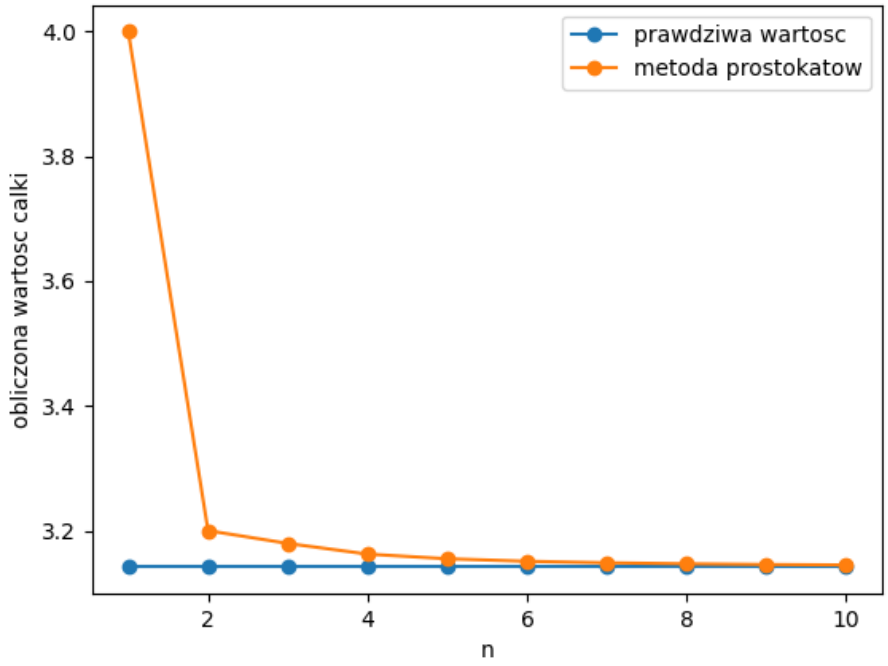
Wszystkie 3 metody zostały przetestowane dla tych samych ilości przedziałów:

$$n \epsilon \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

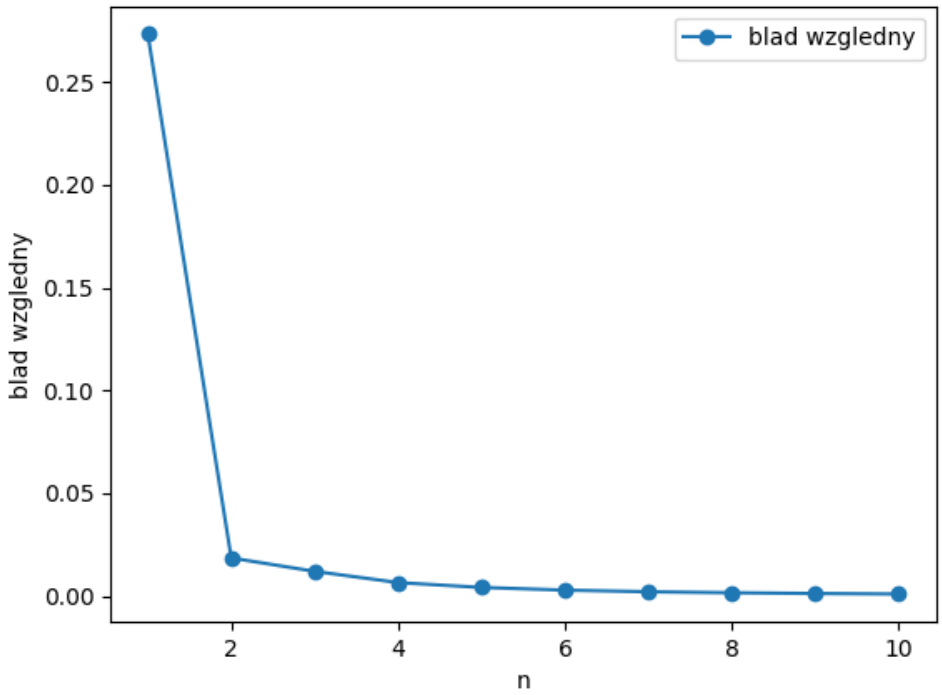
Porównanie wyników 3 metod w zależności od ilości przedziałów:



Porównanie wyników **metody prostokątów** w zależności od ilości przedziałów:



Otrzymany błąd względny **metody prostokątów** w zależności od ilości przedziałów:

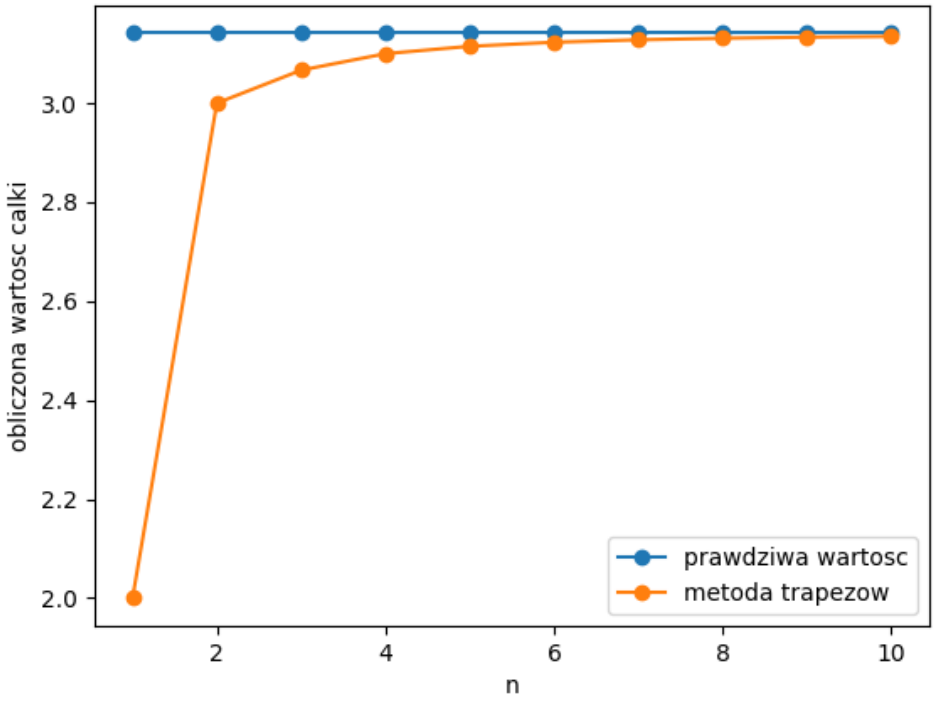


Wartości otrzymanych błędów **metody prostokątów**:

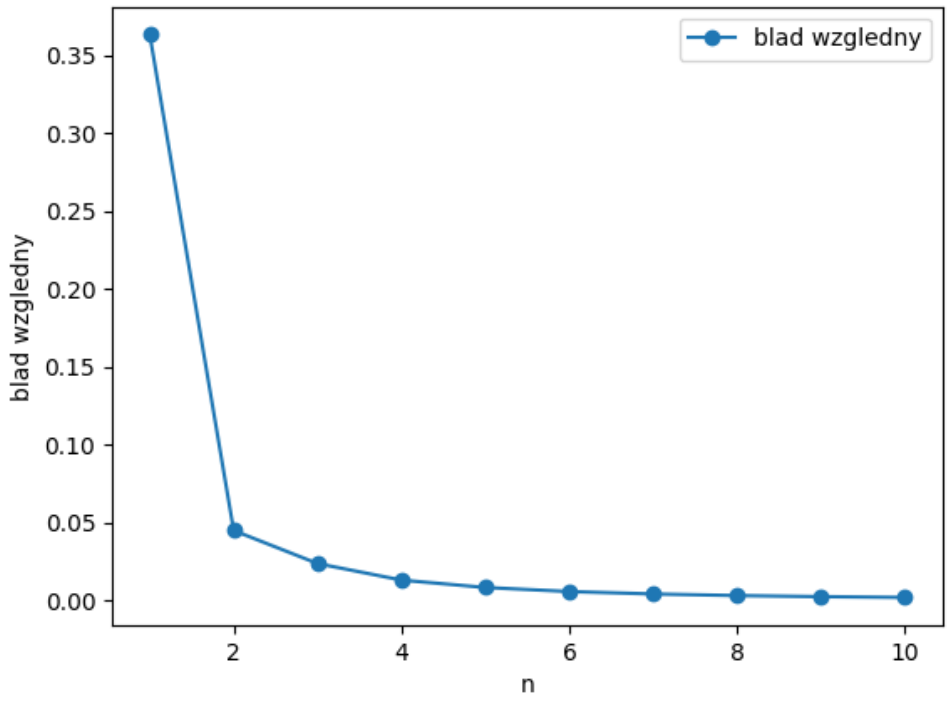
```
====== METODA PROSTOKATOW ======

n = 1   blad = 0.2732395447
n = 2   blad = 0.0185916358
n = 3   blad = 0.0120622022
n = 4   blad = 0.0066082048
n = 5   blad = 0.0042422582
n = 6   blad = 0.0029464534
n = 7   blad = 0.0021650427
n = 8   blad = 0.0016577149
n = 9   blad = 0.0013098436
n = 10   blad = 0.0010609938
```

Porównanie wyników **metody trapezów** w zależności od ilości przedziałów:



Otrzymany błąd względny **metody trapezów** w zależności od ilości przedziałów:

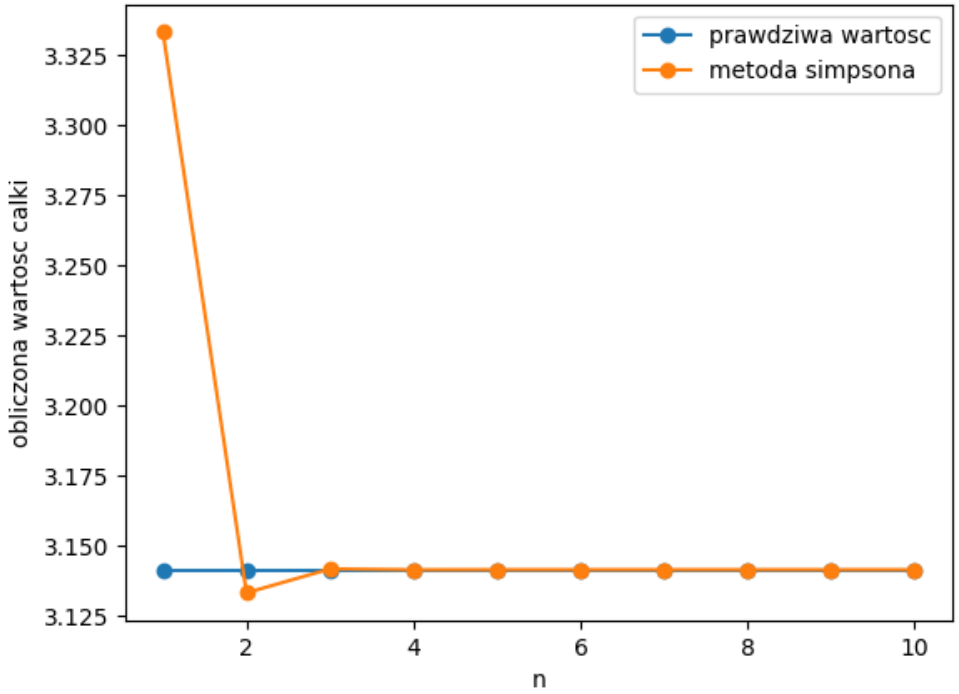


Wartości otrzymanych błędów **metody trapezów**:

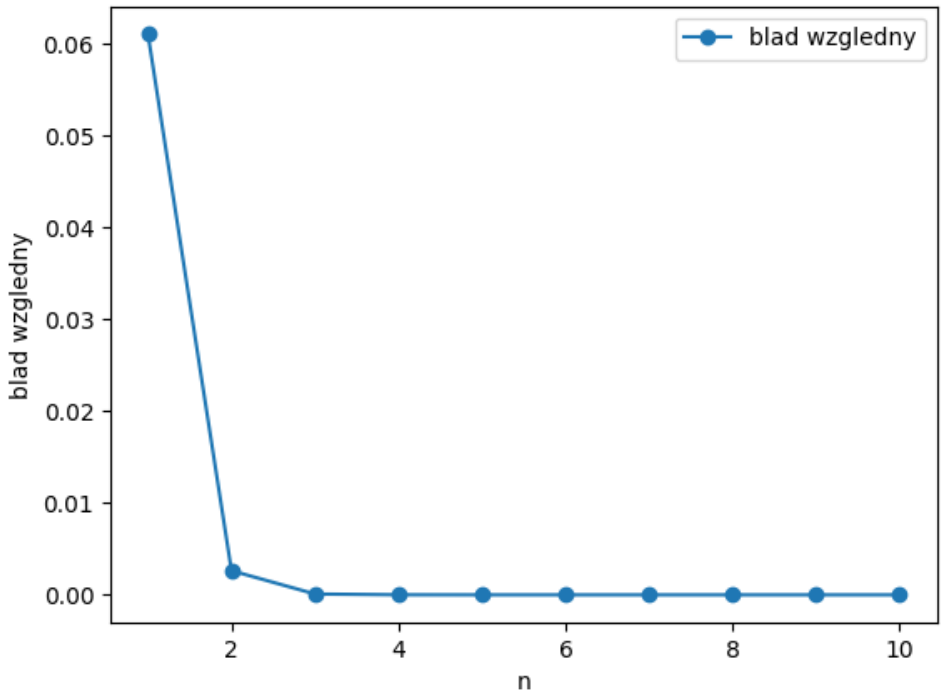
```
====== METODA TRAPEZOW ======

n = 1  blad = 0.3633802276
n = 2  blad = 0.0450703414
n = 3  blad = 0.0238496824
n = 4  blad = 0.0132393528
n = 5  blad = 0.0084863093
n = 6  blad = 0.0058937401
n = 7  blad = 0.0043304054
n = 8  blad = 0.003315574
n = 9  blad = 0.0026197585
n = 10  blad = 0.0021220255
```

Porównanie wyników **metody Simpsona** w zależności od ilości przedziałów:



Otrzymany błąd względny **metody Simpsona** w zależności od ilości przedziałów:



Wartości otrzymanych błędów **metody Simpsona**:

```
====== METODA SIMPSONA ======

n = 1  blad = 0.3633802276
n = 2  blad = 0.0450703414
n = 3  blad = 0.0238496824
n = 4  blad = 0.0132393528
n = 5  blad = 0.0084863093
n = 6  blad = 0.0058937401
n = 7  blad = 0.0043304054
n = 8  blad = 0.003315574
n = 9  blad = 0.0026197585
n = 10  blad = 0.0021220255
```

Wnioski:

- Bez względu na metodę wraz ze wzrostem liczby przedziałów rośnie także dokładność oszacowania
- Najlepiej sprawdza się metoda Simpsona, która już dla małej liczby przedziałów daje wynik stosunkowo dokładny.

Wykorzystany kod w języku Python:

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

# całkowana funkcja
fx = lambda x: 2 / (x**2 + 1)

# przedzial
a = -1
b = 1
length = b - a

# liczby podzialow
ns = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


def metoda_prostokatow(n):
    h = length / n
    x = np.linspace(a - (h/2), b - (h/2), num=n+1, endpoint=True)
    #print(x[1::])
    calka = sum(map(lambda x: fx(x) * length / n, x[1::]))
    return calka

def metoda_trapezow(n):
```

```
    h = length / n
    x = np.linspace(a, b, num=n+1, endpoint=True)
    #print(x)
    calka = sum(map(lambda x: (fx(x) + fx(x - h)) * h / 2, x[1::]))
    return calka

def metoda_simpsona(n):
    h = length / n
    x = np.linspace(a, b, num=n+1, endpoint=True)
    #print(x)
    calka = sum(map(lambda x: (fx(x) + 4 * fx(x - (h/2)) + fx(x-h)) * h /
6, x[1::]))
    return calka


x = np.linspace(min(ns), max(ns), len(ns))

fpi = [math.pi] * (len(ns))

fprostokatow = []
for n in ns:
    fprostokatow.append(metoda_prostokatow(n))

ftrapezow = []
for n in ns:
    ftrapezow.append(metoda_trapezow(n))

fsimpsona = []
for n in ns:
    fsimpsona.append(metoda_simpsona(n))


# porownanie wszystkich wykresow
fig, ax = plt.subplots()
ax.plot(x, fpi, label="prawdziwa wartosc", marker='o')
ax.plot(x, fprostokatow, label="metoda prostokatow", marker='o')
ax.plot(x, ftrapezow, label="metoda trapezow", marker='o')
ax.plot(x, fsimpsona, label="metoda simpsona", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("obliczona wartosc calki")
ax.legend()
plt.show()


# porownanie metody prostokatow
print("\n====== METODA PROSTOKATOW ======\n")

blad_prostokaty = []
for i in range(len(ns)):
    blad_prostokaty.append(abs(fprostokatow[i] - fpi[i])/fpi[i])
    print("n =", ns[i], " blad =", round(blad_prostokaty[i], 10))

fig, ax = plt.subplots()
ax.plot(x, fpi, label="prawdziwa wartosc", marker='o')
ax.plot(x, fprostokatow, label="metoda prostokatow", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("obliczona wartosc calki")
ax.legend()
plt.show()

fig, ax = plt.subplots()
```

```
ax.plot(x, blad_prostokaty, label="blad wzgledny", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("blad wzgledny")
ax.legend()
plt.show()


# porownanie metody trapezow
print("\n====== METODA TRAPEZOW ======\n")

blad_trapezy = []
for i in range(len(ns)):
    blad_trapezy.append(abs(ftrapezow[i] - fpi[i])/fpi[i])
    print("n =", ns[i], " blad =", round(blad_trapezy[i], 10))

fig, ax = plt.subplots()
ax.plot(x, fpi, label="prawdziwa wartosc", marker='o')
ax.plot(x, ftrapezow, label="metoda trapezow", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("obliczona wartosc calki")
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(x, blad_trapezy, label="blad wzgledny", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("blad wzgledny")
ax.legend()
plt.show()


# porownanie metody simpsona
print("\n====== METODA SIMPSONA ======\n")

blad_simpson = []
for i in range(len(ns)):
    blad_simpson.append(abs(fsimpsona[i] - fpi[i])/fpi[i])
    print("n =", ns[i], " blad =", round(blad_trapezy[i], 10))

fig, ax = plt.subplots()
ax.plot(x, fpi, label="prawdziwa wartosc", marker='o')
ax.plot(x, fsimpsona, label="metoda simpsona", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("obliczona wartosc calki")
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(x, blad_simpson, label="blad wzgledny", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("blad wzgledny")
ax.legend()
plt.show()
```

Zadanie 2.

Oblicz wartość całki

$$\int_{-1}^{1} \frac{2}{1+x^2} dx$$

metoda Gaussa-Legendre'a.

Narysuj wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej, $n + 1$.

Przybliżona wartość całki wyraża się wzorem:

$$\int_{a}^{b} f(x)dx = \sum_{i=1}^{n} c_i * f(x_i)$$

Współczynniki c_i są rozwiązaniem równania:

$$\begin{bmatrix} P_0(x_1) & \cdots & P_0(x_n) \\ \vdots & \ddots & \vdots \\ P_{n-1}(x_1) & \cdots & P_{n-1}(x_n) \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} \int_a^b w(x)P_0(x)dx \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Natomiast $x_1, x_2 \ldots x_n$ są miejscami zerowymi wielomianu $P_{n-1}(x_n)$.

W naszym przypadku wykorzystujemy wielomiany Legendre'a i funkcję wagową $w(x) = 1$

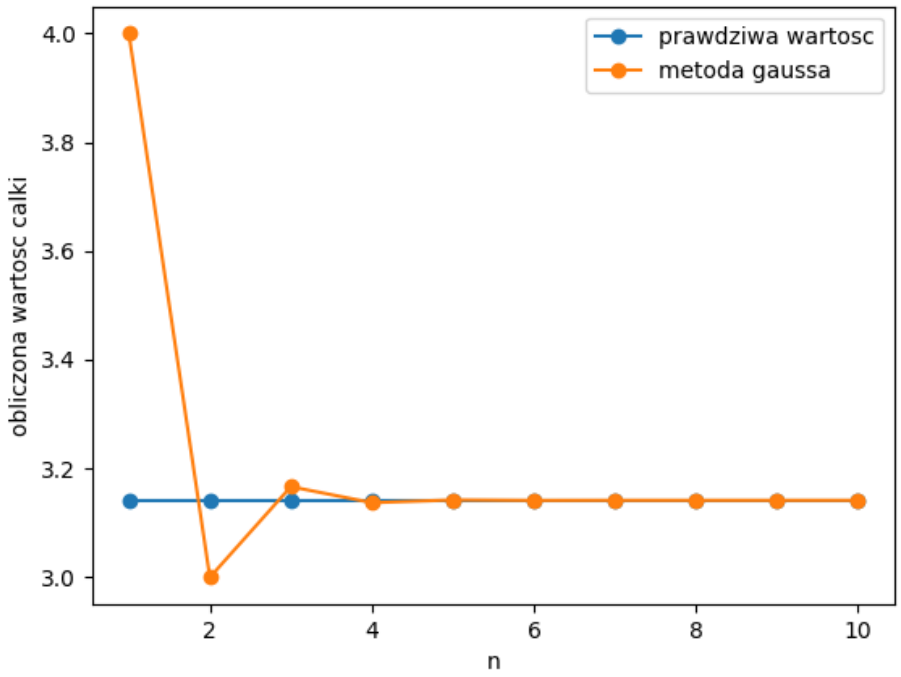
Wykorzystujemy również fakt, że

$$\int_{-1}^{1} w(x)P_0(x)dx = \int_{-1}^{1} 1 \cdot 1 dx = 2$$

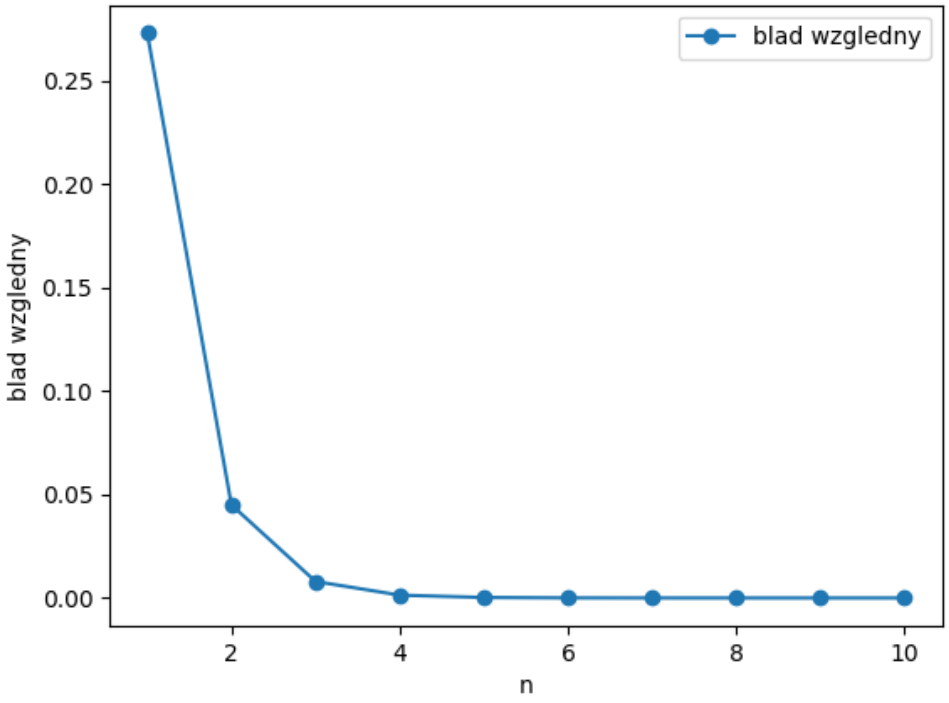
Metoda została przetestowana dla różnych ilości przedziałów:

$$n \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Porównanie wyników **metody Gaussa-Legendre'a** w zależności od ilości przedziałów:



Otrzymany błąd względny **metody Gaussa-Legendre'a** w zależności od ilości przedziałów:



Wartości otrzymanych błędów **metody Gaussa-Legendre'a**:

```
n = 1  blad = 0.2732395447
n = 2  blad = 0.0450703414
n = 3  blad = 0.0079813062
n = 4  blad = 0.0013807492
n = 5  blad = 0.0002386333
n = 6  blad = 4.1138e-05
n = 7  blad = 7.0834e-06
n = 8  blad = 1.2186e-06
n = 9  blad = 2.095e-07
n = 10  blad = 3.6e-08
```

Wnioski:

- Podobnie jak przy poprzednich metodach wraz ze wzrostem liczby przedziałów rośnie także dokładność oszacowania.
- Metoda Gaussa-Legendre'a sprawdza się najlepiej ze wszystkich testowanych metod. Przy n > 5 błąd staje się praktycznie pomijalny.

Wykorzystany kod w języku Python:

```python
import numpy as np
import math
from numpy.polynomial.polynomial import Polynomial
import matplotlib.pyplot as plt

# całkowana funkcja
fx = lambda x: 2 / (x**2 + 1)

# przedzial
a = -1
b = 1
length = b - a

# liczby podzialow
ns = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
#nss = 10

# wyznaczenie wielomianow Legendre'a
def Legendre_polymonials(n):

    Px = Polynomial((0, 1)) # wielomian R = x potrzebny do mnożenia z
wielomianami Legendre'a

    P0 = Polynomial((1))
    P1 = Polynomial((0, 1))
```

```
    polynomials = []
    polynomials.append(P0)
    polynomials.append(P1)

    for n in range(2, n+1):
        Pn = (((2 * n - 1) * Px * polynomials[n-1]) - ((n - 1) *
polynomials[n-2])) / n
        polynomials.append(Pn)

    return polynomials

fgaussa = []

for n in ns:
    polymonials = Legendre_polymonials(n)
    xs = polymonials[n].roots()
    #print(xs)

    A = []

    for i in range(n):
        a = []
        Pi = polymonials[i]
        for j in range(n):
            x = xs[j]
            a.append(Pi(x))
        A.append(a)

    A = np.array(A)

    B = np.zeros(n)
    B[0] = 2

    cs = np.linalg.solve(A, B)
    #print(cs)

    result = 0
    for i in range(n):
        result += cs[i] * fx(xs[i])

    #print(result)
    fgaussa.append(result)


x = np.linspace(min(ns), max(ns), len(ns))

fpi = [math.pi] * (len(ns))

# porownanie wynikow
fig, ax = plt.subplots()
ax.plot(x, fpi, label="prawdziwa wartosc", marker='o')
ax.plot(x, fgaussa, label="metoda gaussa", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("obliczona wartosc calki")
ax.legend()
plt.show()

# porownananie bledow
blad_gauss = []
for i in range(len(ns)):
    blad_gauss.append(abs(fgaussa[i] - fpi[i])/fpi[i])
```

```
    print("n =", ns[i], " blad =", round(blad_gauss[i], 10))

fig, ax = plt.subplots()
ax.plot(x, blad_gauss, label="blad wzgledny", marker='o')
ax.set_xlabel("n")
ax.set_ylabel("blad wzgledny")
ax.legend()
plt.show()
```