Artur Gęsiarz,
Kwiecień 27, 2024

Laboratorium nr 7
MOwNiT – Kwadratury adaptacyjne

1. Treść zadania

1.1. Zadanie pierwsze

Oblicz wartość całki z poprzedniego laboratorium

$$\int_0^1 \frac{4}{1+x^2}\,dx$$

korzystając z kwadratur adaptacyjnych trapezów oraz kwadratur adaptacyjnych Gaussa-Kronroda.

Dla każdej metody narysuj wykres wartości bezwzględnej błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej. Wyniki dodaj do wykresu uzyskanego w poprzednim laboratorium. Przydatna będzie funkcja scipy.integrate.quad vec. Na liczbę ewaluacji funkcji podcałkowej można wpływać pośrednio, zmieniając wartość dopuszczalnego błędu (tolerancji). Przyjmij wartości tolerancji z zakresu od 10^0 do 10^{-14}. Liczba ewalulacji funkcji pod- całkowej zwracana jest w zmiennej info[’neval’].

1.2. Zadanie drugie

Powtórz obliczenia z poprzedniego oraz dzisiejszego laboratorium dla całki

$$\int_0^1 \sqrt{x}\log x\,dx = -\frac{4}{9}$$

oraz całki

$$\int_0^1 \left(\frac{1}{(x-0.3)^2+0.001}+\frac{1}{(x-0.9)^2+0.004}-6\right)dx$$

$$=$$

$$\frac{1}{\sqrt{0.001}}\left(arctg\,\frac{1-0.3}{\sqrt{0.001}}+arctg\,\frac{0.3}{\sqrt{0.001}}\right)+\frac{1}{\sqrt{0.004}}\left(arctg\,\frac{1-0.9}{\sqrt{0.004}}+arctg\,\frac{0.9}{0.004}\right)-6$$

2. Rozwiązanie zadań

2.1. Funkcja podcałkowa dla zadania 1

```
def f1(x):
    return 4 / (1 + x ** 2)
```

2.2. Funkcja podcalkowa dla zadania 2

```
def f2(x):
    return np.where(x > 0, np.sqrt(x) * np.log(x), 0)
```

```
def f3(x, a = 0.001, b = 0.004):
    return 1 / ((x - 0.3) ** 2 + a) + 1 / ((x - 0.9) ** 2 + b) - 6
```

2.3. Zakres tolerancji

```
tolerances = np.logspace(0, -14, num=15)
```

2.4. Przedział całkowania

```
a = 0
b = 1
```

2.5. Prawdziwa wartość całek

```
exact_value_f1 = np.pi
exact_value_f2 = - 4 / 9
exact_value_f3 = 1 / np.sqrt(0.001) * (np.arctan((1 - 0.3) / np.sqrt(0.001)) + np.arctan(0.3 / np.sqrt(0.001))) +
 / np.sqrt(0.004) * (np.arctan((1 - 0.9) / np.sqrt(0.004)) + np.arctan(0.9 / np.sqrt(0.004))) - 6
```

2.6. Inicializacja tablicy błędu względnego

2.6.1. Funkcja podcałkowa pierwsza

```
errors_trap_f1 = []
errors_gauss_kronrod_f1 = []

errors_trap_not_adaptive_f1 = []
errors_mid_not_adaptive_f1 = []
errors_simp_not_adaptive_f1 = []
errors_gauss_not_adaptive_f1 = []
```

2.6.2. Funkcja podcałkowa druga

```
errors_trap_f2 = []
errors_gauss_kronrod_f2 = []

errors_trap_not_adaptive_f2 = []
errors_mid_not_adaptive_f2 = []
errors_simp_not_adaptive_f2 = []
errors_gauss_not_adaptive_f2 = []
```

2.6.3. Funkcja podcałkowa trzecia

```
errors_trap_f3 = []
errors_gauss_kronrod_f3 = []

errors_trap_not_adaptive_f3 = []
errors_mid_not_adaptive_f3 = []
errors_simp_not_adaptive_f3 = []
errors_gauss_not_adaptive_f3 = []
```

2.7. Iniclizacja tablicy liczby ewaluacji w zależności od metody oraz funkcji

2.7.1. Metody nie adaptacyjne

```
evals_not_adaptive = []
```

2.7.2. Funkcja podcalkowa pierwsza

```
evals_trap_f1 = []
evals_gauss_kronrod_f1 = []
```

2.7.3. Funkcja podcałkowa druga

```
evals_trap_f2 = []
evals_gauss_kronrod_f2 = []
```

2.7.4. Funkcja podcałkowa trzecia

```
evals_trap_f3 = []
evals_gauss_kronrod_f3 = []
```

2.8. Funkcja obliczająca całke metodą trapezów z podaną tolerancja

```
def adaptive_trapezoidal_integration(f, a, b, tol):
    integral, err, info = quad_vec(f, a, b, epsabs=tol,quadrature='trapezoid', full_output=True)
    return integral, info.neval
```

2.9. Funkcja obliczające całke metodą Gaussa-Kronroda z podana tolerancja

```
def adaptive_gauss_kronrod_integration(f, a, b, tol):
    integral, err, info = quad_vec(f, a, b, epsabs=tol, full_output=True)
    return integral, info.neval
```

2.10. Funkcja obliczajca całkę metodą Gaussa-Legrande'a

```
def gauss_legendre_integration(n, f):
    nodes, weights = roots_legendre(n)

    x = 0.5 * (nodes + 1)
    w = 0.5 * weights

    integral_value = np.sum(w * f(x))

    return integral_value
```

2.11. Funkcja pomocnicza obliczająca błąd względny oraz liczbę ewaluacji funkcji dla metod adaptacyjnych

```
def calculate_errors_and_evals_for_adaptive(f, method, exact_value, a, b, erros, evals):

    for tol in tolerances:
        integrate, neval = method(f, a, b, tol)
        error = np.abs(np.abs(integrate - exact_value) / exact_value)

        erros.append(error)
        evals.append(neval)

    return erros, evals
```

2.12. Funkcja pomocnicza obliczająca błąd względny oraz liczbę ewaluacji funkcji dla metod nieadaptacyjnych

```
def calculate_errors_and_evals_for_non_adaptive(f, x, m, exact_value):

    num_nodes = 2 ** m + 1

    ### Metoda nieadaptacyjna prostokatow
    dx = (1 - 0) / num_nodes
    x_mid = (x[:-1] + x[1:]) / 2
    integrate = np.sum(f(x_mid) * dx)
    error_mid = np.abs(np.abs(integrate - exact_value) / exact_value)

    ### Metoda nieadaptacyjna trapezow
    integrate = trapz(f(x), x)
    error_trap = np.abs(np.abs(integrate - exact_value) / exact_value)

    ### Metoda nieadaptacyjna Simpsona
    integrate = simps(f(x), x)
    error_simp = np.abs(np.abs(integrate - exact_value) / exact_value)

    ### Metoda nieadaptacyjna Gaussa-Lagrange'a
    integrate = gauss_legendre_integration(m, f)
    error_gauss = np.abs(np.abs(integrate - exact_value) / exact_value)

    return error_mid,error_trap, error_simp, error_gauss
```

2.13. Obliczenie wartości całki dla wszystkich funkcji stosując kwadratury adaptacyjnyne

```
### Metoda adaptacyjna trapezow

errors_trap_f1 ,evals_trap_f1 = calculate_errors_and_evals_for_adaptive(f1, adaptive_trapezoidal_integration,
 exact_value_f1, a, b, errors_trap_f1, evals_trap_f1)

errors_trap_f2 ,evals_trap_f2 = calculate_errors_and_evals_for_adaptive(f2, adaptive_trapezoidal_integration,
 exact_value_f2, a, b, errors_trap_f2, evals_trap_f2)

errors_trap_f3 ,evals_trap_f3 = calculate_errors_and_evals_for_adaptive(f3, adaptive_trapezoidal_integration,
 exact_value_f3, a, b, errors_trap_f3, evals_trap_f3)

### Metoda adaptacyjne Gausaa-Kronroda

errors_gauss_kronrod_f1,evals_gauss_kronrod_f1 = calculate_errors_and_evals_for_adaptive(f1,
 adaptive_gauss_kronrod_integration, exact_value_f1, a, b, errors_gauss_kronrod_f1, evals_gauss_kronrod_f1)

errors_gauss_kronrod_f2,evals_gauss_kronrod_f2 = calculate_errors_and_evals_for_adaptive(f2,
 adaptive_gauss_kronrod_integration, exact_value_f2, a, b, errors_gauss_kronrod_f2, evals_gauss_kronrod_f2)

errors_gauss_kronrod_f3,evals_gauss_kronrod_f3 = calculate_errors_and_evals_for_adaptive(f3,
 adaptive_gauss_kronrod_integration, exact_value_f3, a, b, errors_gauss_kronrod_f3, evals_gauss_kronrod_f3)
```

2.14. Obliczenie wartości wszystkich funkcji stosując kwadratury nieadaptacyjne

```
for m in np.arange(1, 26):

    # Obliczenie liczby węzłów
    num_nodes = 2 ** m + 1
    evals_not_adaptive.append(num_nodes)

    # Rozmieszczenie równoodległych węzłów
    x = np.linspace(0, 1, num_nodes)

    # Obliczenie bledow dla kazdej funkcji podcalkowej
    error_mid_f1, error_trap_f1, error_simp_f1, error_gauss_f1 = calculate_errors_and_evals_for_non_adaptive(f1, x, m,
     exact_value_f1)
    error_mid_f2, error_trap_f2, error_simp_f2, error_gauss_f2 = calculate_errors_and_evals_for_non_adaptive(f2, x, m,
     exact_value_f2)
    error_mid_f3, error_trap_f3, error_simp_f3, error_gauss_f3 = calculate_errors_and_evals_for_non_adaptive(f3, x, m,
     exact_value_f3)

    # Dodanie do tablicy obliczonych wczesniej bledow
    errors_mid_not_adaptive_f1.append(error_mid_f1)
    errors_mid_not_adaptive_f2.append(error_mid_f2)
    errors_mid_not_adaptive_f3.append(error_mid_f3)
    errors_trap_not_adaptive_f1.append(error_trap_f1)
    errors_trap_not_adaptive_f2.append(error_trap_f2)
    errors_trap_not_adaptive_f3.append(error_trap_f3)
    errors_simp_not_adaptive_f1.append(error_simp_f1)
    errors_simp_not_adaptive_f2.append(error_simp_f2)
    errors_simp_not_adaptive_f3.append(error_simp_f3)
    errors_gauss_not_adaptive_f1.append(error_gauss_f1)
    errors_gauss_not_adaptive_f2.append(error_gauss_f2)
    errors_gauss_not_adaptive_f3.append(error_gauss_f3)
```

2.15. Rysowanie wykresu dla funkcji pocałkowej nr. 1

```
plt.figure(figsize=(10, 6))

plt.semilogy(evals_trap_f1, errors_trap_f1, label='Metoda adaptacyjna trapezow')
plt.semilogy(evals_gauss_kronrod_f1, errors_gauss_kronrod_f1, label='Metoda adaptacyjna Gaussa-Kronrod')

plt.semilogy(evals_not_adaptive, errors_trap_not_adaptive_f1, label='Metoda nieadaptacyjna trapezow')
plt.semilogy(evals_not_adaptive, errors_mid_not_adaptive_f1, label='Metoda nieadaptacyjna prostokatow')
plt.semilogy(evals_not_adaptive, errors_simp_not_adaptive_f1, label='Metoda nieadaptacyjna Simpsona')
plt.semilogy(evals_not_adaptive, errors_gauss_not_adaptive_f1, label='Metoda nieadaptacyjna Gaussa-Lagrenge\'a')

plt.title('Błąd bezwzględny w zależności od liczby ewaluacji funkcji podcałkowej')
plt.xlabel('Liczba ewaluacji funkcji podcałkowej')
plt.ylabel('Błąd bezwzględny')
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```

2.16. Rysowanie wykresu dla funkcji podcałkowej nr. 2

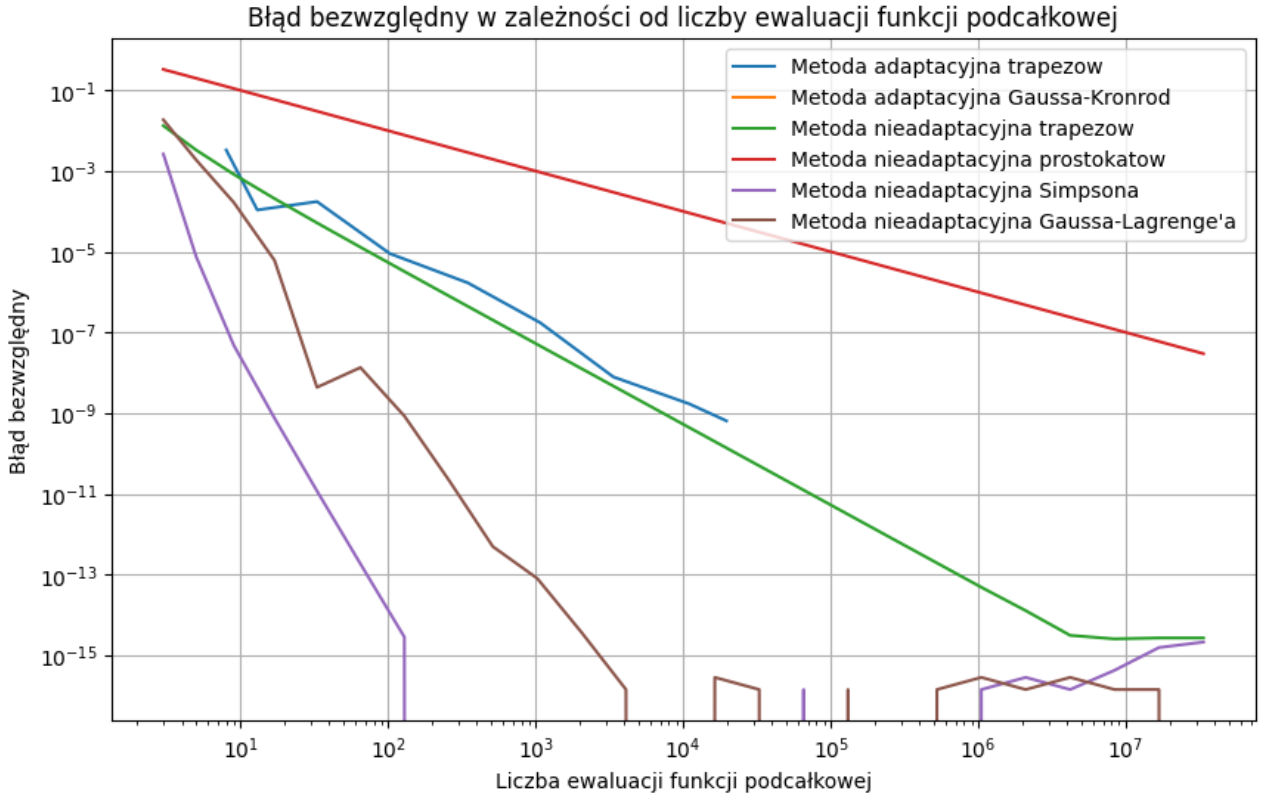
```
plt.figure(figsize=(10, 6))

plt.semilogy(evals_trap_f2, errors_trap_f2, label='Metoda adaptacyjna trapezow')
plt.semilogy(evals_gauss_kronrod_f2, errors_gauss_kronrod_f2, label='Metoda adaptacyjna Gaussa-Kronrod')

plt.semilogy(evals_not_adaptive, errors_trap_not_adaptive_f2, label='Metoda nieadaptacyjna trapezow')
plt.semilogy(evals_not_adaptive, errors_mid_not_adaptive_f2, label='Metoda nieadaptacyjna prostokatow')
plt.semilogy(evals_not_adaptive, errors_simp_not_adaptive_f2, label='Metoda nieadaptacyjna Simpsona')
plt.semilogy(evals_not_adaptive, errors_gauss_not_adaptive_f2, label='Metoda nieadaptacyjna Gaussa-Lagrenge\'a')

plt.title('Błąd bezwzględny w zależności od liczby ewaluacji funkcji podcałkowej')
plt.xlabel('Liczba ewaluacji funkcji podcałkowej')
plt.ylabel('Błąd bezwzględny')
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```

2.17. Rysowanie wykresu dla funkcji podcałkowej nr. 3

```
plt.figure(figsize=(10, 6))

plt.semilogy(evals_trap_f3, errors_trap_f3, label='Metoda adaptacyjna trapezow')
plt.semilogy(evals_gauss_kronrod_f3, errors_gauss_kronrod_f3, label='Metoda adaptacyjna Gaussa-Kronrod')

plt.semilogy(evals_not_adaptive, errors_trap_not_adaptive_f3, label='Metoda nieadaptacyjna trapezow')
plt.semilogy(evals_not_adaptive, errors_mid_not_adaptive_f3, label='Metoda nieadaptacyjna prostokatow')
plt.semilogy(evals_not_adaptive, errors_simp_not_adaptive_f3, label='Metoda nieadaptacyjna Simpsona')
plt.semilogy(evals_not_adaptive, errors_gauss_not_adaptive_f3, label='Metoda nieadaptacyjna Gaussa-Lagrenge\'a')

plt.title('Błąd bezwzględny w zależności od liczby ewaluacji funkcji podcałkowej')
plt.xlabel('Liczba ewaluacji funkcji podcałkowej')
plt.ylabel('Błąd bezwzględny')
plt.xscale('log')
plt.yscale('log')
plt.legend()
plt.grid(True)
plt.show()
```

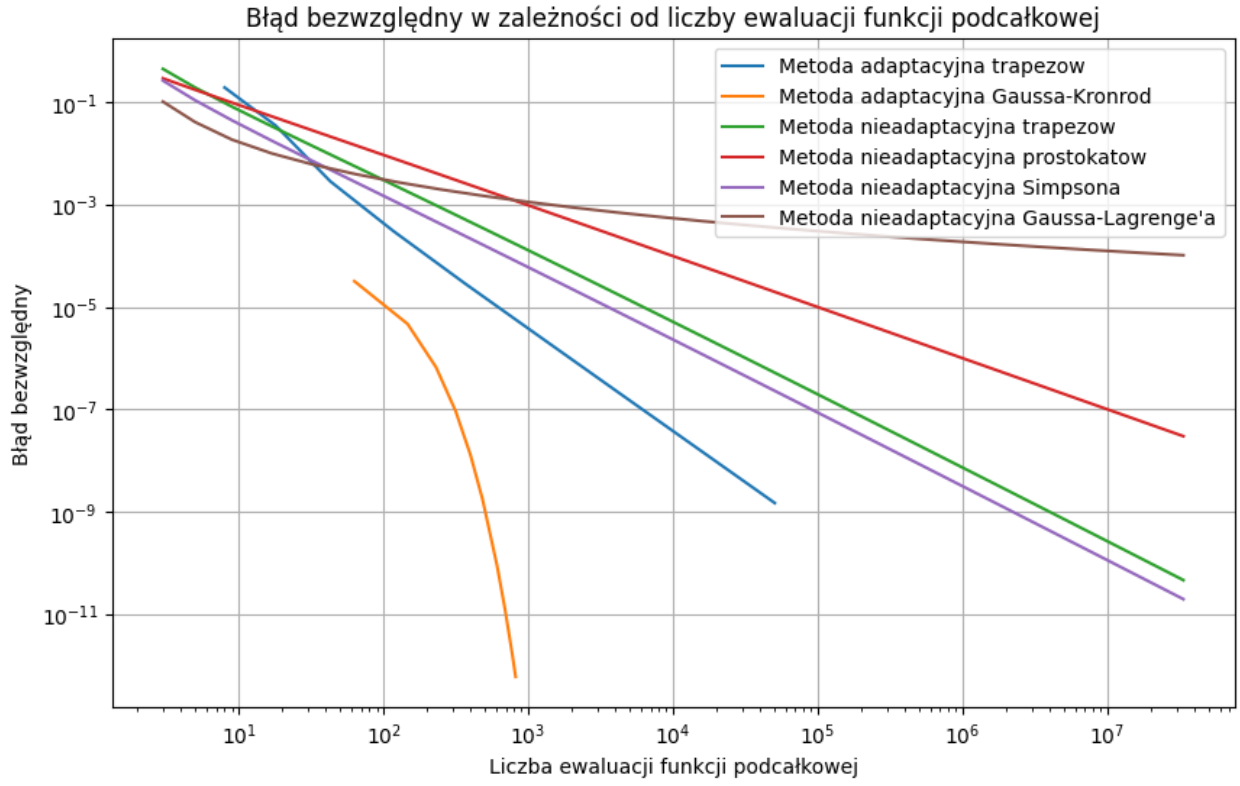
3. Wykres

3.1. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej nr. 1



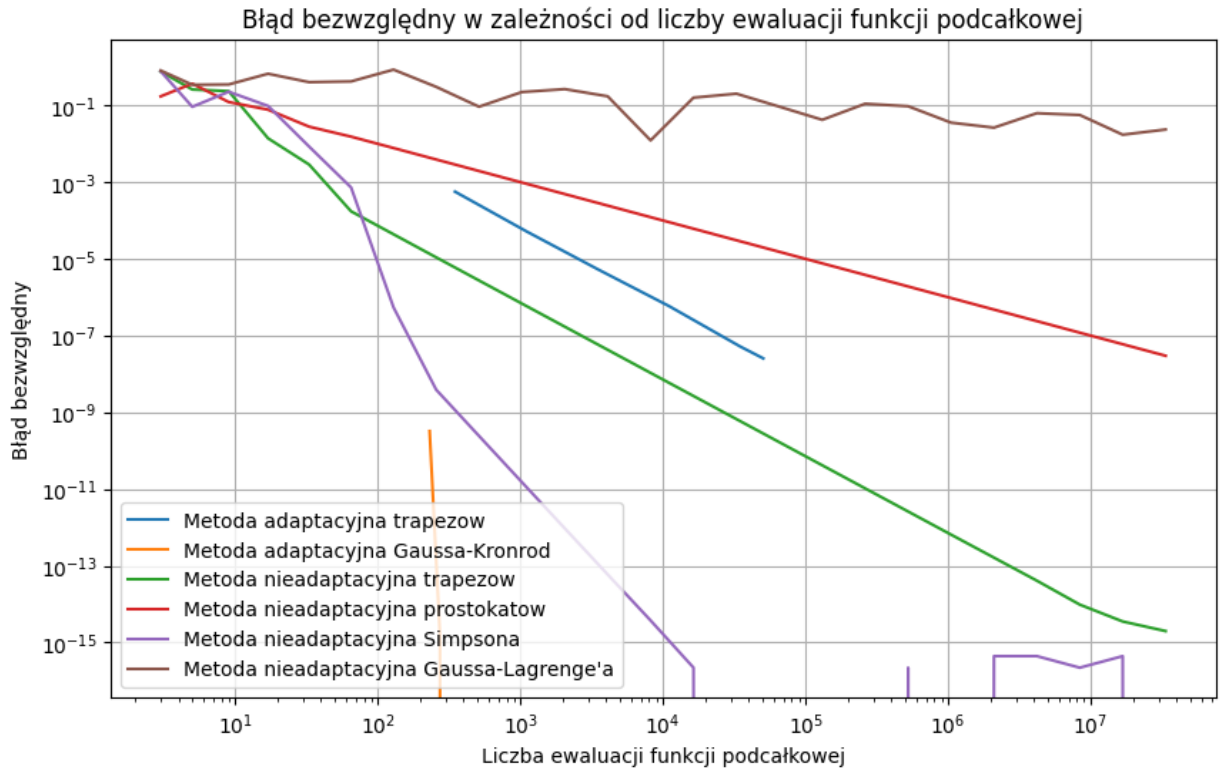
Wykres 1. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej

3.2. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej nr. 2



Wykres 2. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej

3.3. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej nr. 3



Wykres 3. Wykres błędu względnego w zależności od liczby ewaluacji funkcji podcałkowej

4. Wnioski

   Metody adaptacyjne, zarówno trapezów, jak i Gaussa-Kronroda, osiągnęły wysoką dokładność w obliczaniu wartości całek dla wszystkich trzech funkcji podcałkowych.

   Przy bardzo niskich wartościach tolerancji (rzędu 10^{-14}), obie metody uzyskały wyniki bliskie rzeczywistym wartościom całek z bardzo małym błędem względnym.

   Jak można się było spodziewać, zmniejszanie tolerancji (tj. wymaganie większej dokładności) prowadziło do wzrostu liczby ewaluacji funkcji podcałkowej.

   Metoda Gaussa-Kronroda, będąca bardziej zaawansowaną techniką, wymagała generalnie mniejszej liczby ewaluacji dla osiągnięcia podobnych poziomów dokładności w porównaniu do metody trapezów.

   Metoda Gaussa-Kronroda okazała się bardziej efektywna pod względem liczby ewaluacji funkcji, szczególnie przy niższych wartościach tolerancji. Dla większości przypadków, metoda Gaussa-Kronroda osiągała mniejszy błąd względny przy mniejszej liczbie ewaluacji w porównaniu do metody trapezów.

   Metoda trapezów jest prostsza i może być bardziej intuicyjna w implementacji, ale przy wymaganiach większej dokładności staje się mniej efektywna.

   Dla funkcji bardziej złożonych lub posiadających osobliwości, jak w przypadku trzeciej funkcji podcałkowej, metody adaptacyjne wykazywały swoją przewagę, dostosowując gęstość siatki punktów całkowania w obszarach wymagających większej precyzji.

   Funkcja druga, zawierająca logarytm i pierwiastek, była trudniejsza do całkowania, co skutkowało większymi różnicami w liczbie ewaluacji i błędzie względnym pomiędzy metodami.

5. Bibliografia
   Wykład MOwNiT - prowadzony przez dr. Inż. K. Rycerz
   Prezentacje – dr. Inż. M. Kuta

6. Dodatkowe informacje
   Rozwiązanie obu zadań znajduje się odpowiednio w pliku ex1_ex2.ipynb.