Laboratorium 07

Kwadratury adaptacyjne

Iga Antonik, Helena Szczepanowska

Zadanie 1

Zadanie 1. Oblicz wartość całki z poprzedniego laboratorium

$$\int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x\,, \tag{1}$$

korzystając z:

(a) kwadratur adaptacyjnych trapezów,

(b) 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 podcałkowej zwracana jest w zmiennej `info['neval']`.

Rozwiązanie

Biblioteki

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import trapz, simps, quad, quad_vec
```

Definicja funkcji podcałkowej

In [ ]:
```
def f(x):
    return 4 / (1 + x**2)
```

Funkcja obliczająca wartość błędu względnego

In [ ]:
```
def relative_error(exact, approx):
    return np.abs((exact - approx) / exact)
```

Obliczanie wartości całek metodą trapezów, Simpsona oraz punktu środkowego

In [ ]:
```
# Zakres wartości m
m_values = np.arange(1, 26)

# Listy przechowujące błędy dla każdej metody
errors_trapezoidal = []
errors_simpson = []
errors_midpoint = []
integral = []
# Pętla obliczająca wartości całek i błędów dla różnych wartości m
for m in m_values:
    n = 2 ** m + 1
    x = np.linspace(0, 1, n)
    y = f(x)

    # Metoda trapezów
    integral_trapezoidal = trapz(y, x)
```

```
    error_trapezoidal = relative_error(np.pi, integral_trapezoidal)
    errors_trapezoidal.append(error_trapezoidal)

    # Metoda Simpsona
    integral_simpson = simps(y, x)
    error_simpson = relative_error(np.pi, integral_simpson)
    errors_simpson.append(error_simpson)

    # Metoda punktu środkowego
    x_mid = (x[1:] + x[:-1]) / 2
    y_mid = f(x_mid)
    integral_midpoint = np.sum(y_mid * (x[1:] - x[:-1]))
    error_midpoint = relative_error(np.pi, integral_midpoint)
    errors_midpoint.append(error_midpoint)

print('Final integral values: ', integral_trapezoidal,integral_simpson,integral_midpoint)

n_values = 2**m_values +1
```

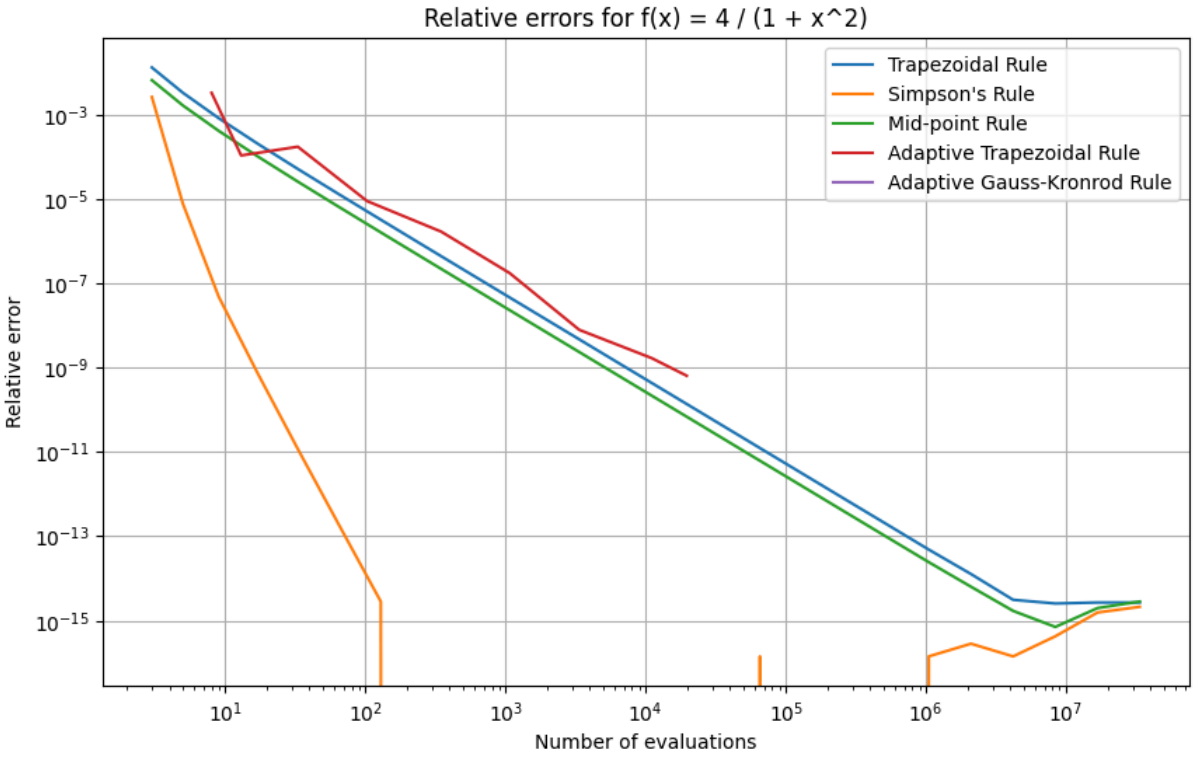
```
Final integral values:  3.1415926535898016 3.1415926535897865 3.141592653589802
```

Obliczenie wartości całek dla metod adaptacyjnych trapezów i Gaussa-Kronroda

In [ ]:
```
a, b = 0, 1
tolerances = np.logspace(0, -14, num=15, base=10)
results_trap = []
results_gauss = []
exact_value = np.pi

for tol in tolerances:

    integral_trap, err, info = quad_vec(f, a, b, epsabs=tol, norm='max', quadrature='trapezoid', full_output=Tr
    relative_trap = relative_error(exact_value, integral_trap)
    results_trap.append((tol, info.neval, relative_trap, integral_trap))

    integral_gauss, err, info = quad_vec(f, a, b, epsabs=tol, norm='max', quadrature='gk21', full_output=True)
    relative_gauss = relative_error(exact_value, integral_gauss)
    results_gauss.append((tol, info.neval, relative_gauss, integral_gauss))

evals_trap = [result[1] for result in results_trap]
errors_trap = [result[2] for result in results_trap]

evals_gauss = [result[1] for result in results_gauss]
errors_gauss = [result[2] for result in results_gauss]

integral_trap = [result[3] for result in results_trap]
integral_gauss = [result[3] for result in results_gauss]
```

Wykres

In [ ]:
```
plt.figure(figsize=(10, 6))
plt.plot(n_values, errors_trapezoidal, label='Trapezoidal Rule')
plt.plot(n_values, errors_simpson, label='Simpson\'s Rule')
plt.plot(n_values, errors_midpoint, label='Mid-point Rule')
plt.plot(evals_trap, errors_trap, label='Adaptive Trapezoidal Rule')
plt.plot(evals_gauss, errors_gauss, label='Adaptive Gauss-Kronrod Rule')
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Number of evaluations')
plt.ylabel('Relative error')
plt.title('Relative errors for f(x) = 4 / (1 + x^2)')
plt.legend()
plt.grid(True)
plt.show()
```



Wnioski

Dla zadanej funkcji metoda kwadratur adaptacyjnych trapezów zwiększa swoją dokładność wraz ze wzrostem liczby ewaluacji funkcji co jest efektem pożądanym, jednak w tym przypadku nie jest ona az tak dokładna jak metoda punktu środkowego oraz trapezów. Spośród wszystkich metod najbardziej dokładna jest metoda Gaussa-Kronroda, poniewaz w przypadku tej metody błąd nie występuje wcale niezaleznie od liczby ewaluacji funkcji.

Zadanie 2

Zadanie 2. Powtórz obliczenia z poprzedniego oraz dzisiejszego laboratorium dla całek

(a)

$$\int_0^1 \sqrt{x} \log x \, \mathrm{d}x = -\frac{4}{9}, \tag{2}$$

(b)

$$\int_0^1 \left(\frac{1}{(x-0.3)^2 + a} + \frac{1}{(x-0.9)^2 + b} - 6 \right) \mathrm{d}x \,. \tag{3}$$

We wzorze (3) przyjmij $a = 0.001$ oraz $b = 0.004$. Błąd kwadratury dla całki (3) oblicz, wykorzystując fakt, że

$$\int_0^1 \frac{1}{(x-x_0)^2 + a} \, \mathrm{d}x = \frac{1}{\sqrt{a}} \left(\operatorname{arctg} \frac{1-x_0}{\sqrt{a}} + \operatorname{arctg} \frac{x_0}{\sqrt{a}} \right). \tag{4}$$

Rozwiązanie

Funkcje podcałkowe

In [ ]:
```
def f1(x):
    return np.where(x > 0, np.sqrt(x) * np.log(x), 0)
```

```
def f2(x):
    term1 = 1 / ((x - 0.3)**2 + 0.001)
    term2 = 1 / ((x - 0.9)**2 + 0.004)
    return term1 + term2 - 6
```

Funkcja obliczająca faktyczną wartość całki dla f2(x)

In [ ]:
```
def f2_exact_value(x0, x1, a, b):
    term1 = (np.arctan((1 - x0) / np.sqrt(a)) + np.arctan(x0 / np.sqrt(a))) / np.sqrt(a)
    term2 = (np.arctan((1 - x1) / np.sqrt(b)) + np.arctan(x1 / np.sqrt(b))) / np.sqrt(b)
    return term1 + term2 - 6
```

Funkcja obliczająca wartości całek

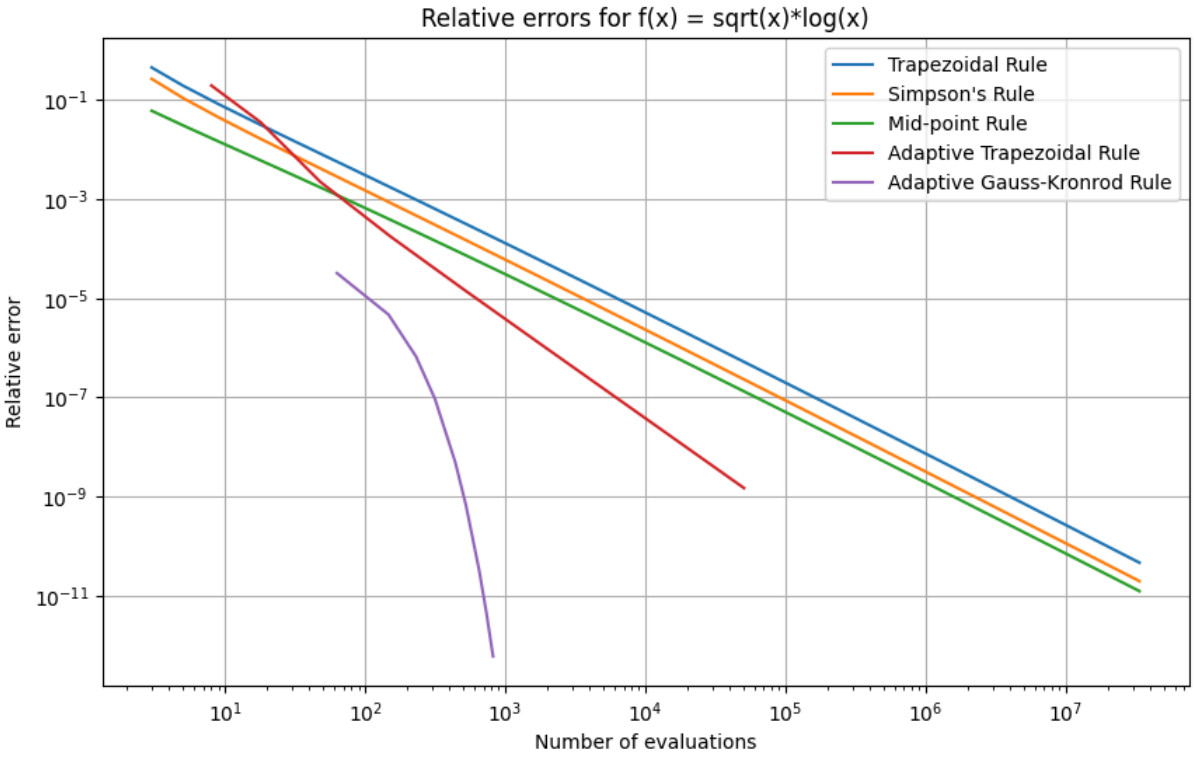
In [ ]:
```
def count_intergrals(f, a, b, exact_value, title):

    # Zakres wartości m
    m_values = np.arange(1, 26)

    # Listy przechowujące błędy dla każdej metody
    errors_trapezoidal = []
    errors_simpson = []
    errors_midpoint = []

    # Pętla obliczająca wartości całek i błędów dla różnych wartości m
    for m in m_values:
        n = 2 ** m + 1
        x = np.linspace(0, 1, n)
        y = f(x)

        # Metoda trapezów
        integral_trapezoidal = trapz(y, x)
        error_trapezoidal = relative_error(exact_value, integral_trapezoidal)
        errors_trapezoidal.append(error_trapezoidal)

        # Metoda Simpsona
        integral_simpson = simps(y, x)
        error_simpson = relative_error(exact_value, integral_simpson)
        errors_simpson.append(error_simpson)

        # Metoda punktu środkowego
        x_mid = (x[1:] + x[:-1]) / 2
        y_mid = f(x_mid)
        integral_midpoint = np.sum(y_mid * (x[1:] - x[:-1]))
        error_midpoint = relative_error(exact_value, integral_midpoint)
        errors_midpoint.append(error_midpoint)


    n_values = 2**m_values +1

    # Wyliczenie wartości całek i błędów dla metod adaptacyjnych trapezów i Gaussa-Kronroda dla tolerancji od 1
    a, b = 0, 1
    tolerances = np.logspace(0, -14, num=14, base=10)
    results_trap = []
    results_gauss = []

    for tol in tolerances:

        integral_trap, err, info = quad_vec(f, a, b, epsabs=tol, norm='max', quadrature='trapezoid', full_outpu
        relative_trap = relative_error(exact_value, integral_trap)
        results_trap.append((tol, info.neval, relative_trap))

        integral_gauss, err, info = quad_vec(f, a, b, epsabs=tol, norm='max', quadrature='gk21', full_output=Tr
        relative_gauss = relative_error(exact_value, integral_gauss)
        results_gauss.append((tol, info.neval, relative_gauss, integral_gauss))

    evals_trap = [result[1] for result in results_trap]
    errors_trap = [result[2] for result in results_trap]

    evals_gauss = [result[1] for result in results_gauss]
    errors_gauss = [result[2] for result in results_gauss]

    print('Final integral values: ', integral_trapezoidal,integral_simpson,integral_midpoint, integral_trap, in

    # Rysowanie wykresu
    plt.figure(figsize=(10, 6))
    plt.plot(n_values, errors_trapezoidal, label='Trapezoidal Rule')
    plt.plot(n_values, errors_simpson, label='Simpson\'s Rule')
    plt.plot(n_values, errors_midpoint, label='Mid-point Rule')
    plt.plot(evals_trap, errors_trap, label='Adaptive Trapezoidal Rule')
    plt.plot(evals_gauss, errors_gauss, label='Adaptive Gauss-Kronrod Rule')
    plt.yscale('log')
```

```
    plt.xscale('log')
    plt.xlabel('Number of evaluations')
    plt.ylabel('Relative error')
    plt.title('Relative errors for ' + title)
    plt.legend()
    plt.grid(True)
    plt.show()
```

Wynik i wykres dla f1(x)

In [ ]:
```
a, b = 0, 1
exact_value_f1 = quad(f1, a, b)[0]
print('Expected value: ', exact_value_f1)
count_intergrals(f1, a, b, exact_value_f1, 'f(x) = sqrt(x)*log(x)' )
```

```
Expected value:  -0.4444444444444475
Final integral values:  -0.4444444444424053 -0.4444444444357836 -0.4444444444498915 -0.4444444437898762 -0.444444
4444447074
```



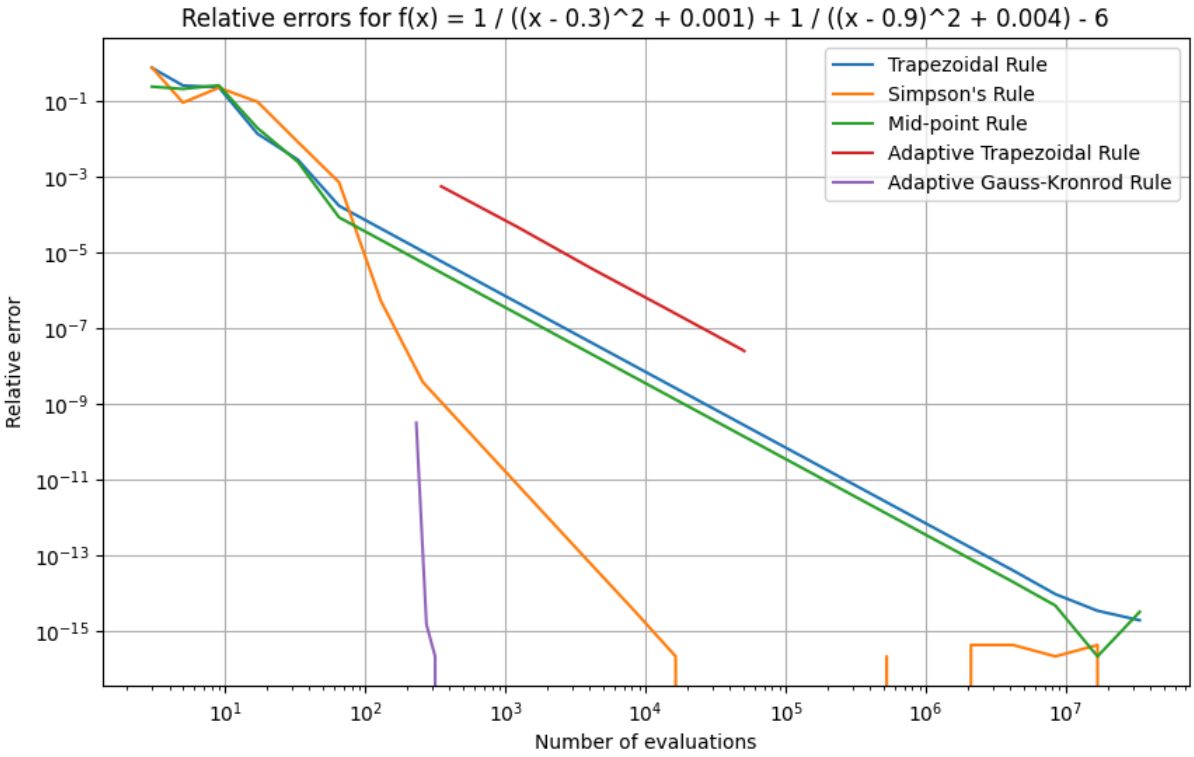
Wnioski

Analizując wyniki oraz wykres dla funkcji f1(x) mozna zobserwować, że metoda kwadratur adaptacyjnych trapezów skutecznie wyznaczyła przybliżoną wartość całki, a jej błąd względny maleje szybciej niż dla pozostałych metod. Jednak najmniejszy błąd bezwzględny osiągneła metoda Gaussa-Kronroda.

Wynik i wykres dla f2(x)

In [ ]:
```
a, b = 0, 1
exact_value_f2 = f2_exact_value(0.3, 0.9, 0.001, 0.004)
print('Expected value: ', exact_value_f2)
count_intergrals(f2, a, b, exact_value_f2, 'f(x) = 1 / ((x - 0.3)^2 + 0.001) + 1 / ((x - 0.9)^2 + 0.004) - 6')
```

```
Expected value:  128.2441502724197
Final integral values:  128.24415027241994 128.2441502724197 128.24415027242011 128.24415353734065 128.244150272
4197
```



Wnioski

Z powyższych wyników oraz wykresu dla funkcji f2(x) wynika, ze wszystkie metody przy odpowiednio dużej liczbie ewaluacji funkcji poprawnie przybliżyły wartość całki, jednak ponownie to metoda Gaussa-Kronroda zwróciła najdokładniejszy wynik. Ze wszystkich obserwacji wynika, ze metoda Gaussa-Kronroda jest najbardziej uniwersalna i dokładna.

Bibliografia

Prezentacja z wykładu nr 5 - kwadratury

https://uwm.edu.pl/wnt/kib/wp-content/uploads/2019/03/08_Ca%C5%82kowanie-numeryczne.ppt

https://home.agh.edu.pl/~funika/mownit/lab5/calkowanie.pdf